THESIS

A PERFORMANCE EVALUATION OF THE COUPLING INFRASTRUCTURE WITHIN THE COMMUNITY EARTH SYSTEM MODEL™

Submitted by

Sheri A. Mickelson

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2018

Master's Committee:

Advisor: Louis-Noel Pouchet

Sanjay Rajopadhye
David Randall

ABSTRACT

A PERFORMANCE EVALUATION OF THE COUPLING INFRASTRUCTURE WITHIN THE COMMUNITY EARTH SYSTEM MODEL™

Earth System models (ESMs) are complex software packages comprised of millions of lines of code used to simulate many different Earth processes. ESMs simulate the dynamical and physical states of the atmosphere, land, ocean, sea ice, rivers, and glaciers and coordinate the interactions between them. Many computational challenges exist within these models and future systems are putting more pressure on these challenges. In order to alleviate some of the pressure, it is important to study the performance challenges that exist within the models in order to understand the optimizations that need to be performed as we move to exascale systems. This work studies the performance of the coupling infrastructure between the modeling components within the Community Earth System Model. The coupler is responsible for the data exchanges between the different modeling components and while it has a small computational footprint, it has the potential to have a large impact on performance if the component resources are dispersed in incorrect proportions. This work explains and addresses this issue and provides easy solutions for users to save thousands of core cpu hours.

ACKNOWLEDGEMENTS

I would like to thank the Computer Science Department at Colorado State University for their support in this work, my advisor Louis-Noel Pouchet, and my Master's committee. I would also like to thank the National Center for Atmospheric Research (NCAR) for the support they have given to me while attending CSU and the support for this work. I would like to thank John Dennis at NCAR for his encouragement, support, and for his advice reviewing this document.

I would like to acknowledge high-performance computing support from Yellowstone provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation.

I would like to acknowledge high-performance computing support from Cheyenne provided by NCAR's Computational and Information Systems Laboratory, sponsored by the National Science Foundation.

DEDICATION

I would like to dedicate this thesis to my family for all of their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 State of Practice

Earth System Models (ESMs) are used to study complex interactions within Earth's systems. They are used to model processes present in the atmosphere, ocean, land, sea ice, rivers, and glaciers. The software behind these models are complex, containing equations to compute dynamics within all of the models, as well as handling the laws of physics.

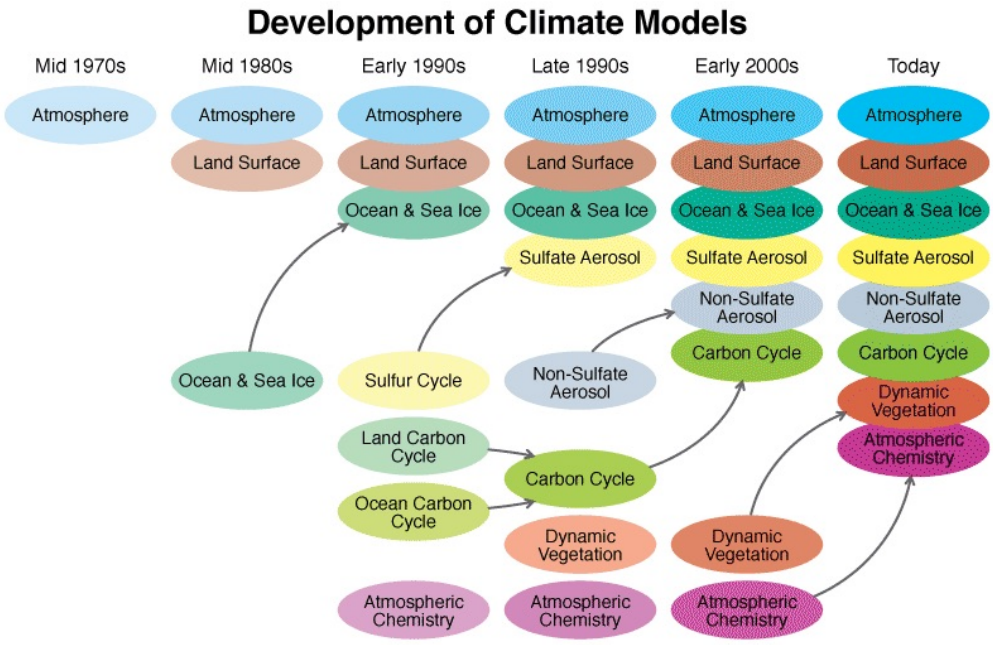


Figure 1.1: The complexity of climate modeling has been increasing for the last forty years [3]. Image credit: https://www.giss.nasa.gov/research/briefs/puma_02

As seen within Figure 1.1, ESMs have been growing more complex as they are able to take advantage of technological advances in computing. During the last forty years, climate science was able to take advantage of clock speeds doubling almost every two years [4] and it has allowed

for more physical processes to be modeled and the simulations to be run at finer resolutions. In order to enable more processes to be modeled, the time is takes to complete a simulation must not degrade. Any degradation in performance increases time to solution and starts to hinder the progress of the science being done. In the past, we've relied on Moore's law to counter the cost of doing the extra calculations, but today we can no longer rely on processor speeds increasing at that same rate. Therefore, to enable more science within the ESMs at a similar cost, we need to take a closer look at model performance, looking for greater concurrency and memory management within the models [5].

We are also seeing new architectures being developed and we must adapt ESMs in oder to take advantage of these new hardware designs [6]. This is a difficult problem because it will require a lot of work to port these models to different architectures and will require even more work to optimize the code once ported.

As part of this effort, it is important to understand the model's performance on current platforms in order to determine the problems they may encounter on these new types of platforms as we approach exascale computing. The work presented here explores the performance within the coupling layer of the well known ESM, the Community Earth System Model (CESM) [7] [8].

CESM is composed of separate model components that simulate the atmosphere, land surface, ocean, sea ice, glaciers, river runoff, and waves. These different model components are connected to each other through a coupling interface. The coupler within CESM coordinates the data transfers within the different component models, coordinates the model time stepping, and synchronizes the running of the model components. While the computational cost of the coupler is small compared to the cost of running the model components, a lot of time can be spent in the coupler within MPI barrier calls if resources are not balanced correctly between the component models. If CESM is to meet the challenges of exascale computing, the performance of the coupler will need to be improved and the idle time caused by the synchronization between the components must be eliminated.

Load balancing CESM in order to reduce coupler idle time is a difficult task because the model can be run in different configurations and each configuration has different performance characteristics. When a scientist sets up an experiment with CESM, it will give each model a default number of resources. Different resolutions, component model combinations, physics options, output frequencies, and different versions of the code base all have an impact on model performance. If the scientists changes any of these from the default, which is usually the case, the model will become imbalanced and will need to be rebalanced by an expert. If this is not done, the model run can be about 50% more expensive to run due to the idle time in the coupler. This work explains and addresses this issue and provides easy solutions for users to save thousands of core cpu hours.

1.2 Contributions

The contributions from this work include the following:

1. A detailed performance analysis of CESM.

2. A detailed performance analysis of CESM in production.

3. The evaluation and validation of optimal model component layouts for CESM.

Chapters 2 and 3 discuss the performance of the CESM modeling components and the coupler. Chapters 4 and 5 discuss the performance analysis of CESM and the coupler in production. Chapter 6 discusses the evaluation and validation of optimal model component layouts and resource allocations. Chapter 7 includes related work in regards to load balancing climate models. Finally Chapter 8 discusses future work that can be done and concluding remarks are noted in Chapter 9.

Chapter 2

Background

Earth System Models (ESMs) are some of the most complex scientific software packages [9] and are amongst the most computationally challenging scientific codes [10]. The size of the source codes of some of the most widely used models from around the world range from forty thousand lines of code all the way to 1.5 million lines of code [9]. While each ESM looks to model the same Earth processes, each vary in the amount of code complexity. For example, some climate models simulate biogeochemistry in the soil or volcanic eruptions, while other models use prescribed values for these processes. The more processes that are derived versus prescribed increases the complexity of the model and increases the cost of running the model.

About every five years, modeling centers from around the world participate in Coupled Model Inter comparison Projects (CMIPs). The goal of these exercises are to have different modeling centers run the same set of experiments in order to get a better understanding of past, present, and future climates. These experiments range from predicting climates under different CO2 mitigation strategies, to the impacts of deforestation, to hind casting paleoclimates. The current iteration, CMIP6 [11], will contain results from about forty-four unique climate models from around the world. While these experiments give valuable scientific information, they can also give us the opportunity to compare the computational performance of different climate models performing the same experiment, providing a fair comparison between them. Balaji, et al. [10] found that the variability in the computational performance of different ESMs is quite large. The performance differences extends from being able to simulate roughly one year of climate per wallclock day all the way to 36 years of climate per wallclock day for the same experiment setup. The variants are a factor of code complexity, differences in the total number of grid points, the computational resources given to the problem (node counts and processor speeds), and the efficiencies of the different codes.

Most coupled climate models suffer from load balance problems where the different component models couple together to pass boundary conditions to each other [10]. Belaji, et al. found that across different climate models, the coupling between different modeling components could be as little as 1% of the total runtime all the way to 62% of the total runtime. The percentage of time includes the time it takes to interpolate between different physics grids and computing fluxes, but it also includes any idle time from load imbalance between the components. The authors note that some of the high coupler costs are likely due to load imbalance and performance improvements could be seen if compute tasks were proportioned differently between the modeling components.

2.1 The Community Earth System Model

CESM version 2 is a general circulation model that is used to predict and understand the Earth's climate [7] [8]. It is composed of separate model components that each model a separate Earth system and each have unique performance characteristics. CESM contains an atmosphere, ocean, land, sea ice, glacier, river runoff, and wave model. All of these models are connected by a coupling interface, where flux values and state information are interpolated and passed between the modeling components. Figure 2.1 shows an overview summary of the different processes that are modeled within CESM.

CESM is one of the most complex climate models of the world [9]. The code for CESM consists of 1.5 million lines of Fortran with a user interface and framework written in Python. CESM uses a hybrid parallelization approach which uses both MPI and OpenMP to handle the parallelization within each model component and for the communication between components. The use of GPU's is currently an exploratory topic and, thus, not available in the current released version of the code.

The CESM user community is very broad and, thus the model is required to run on everything from Supercomputers down to laptops. This requires it to contain different model setups that can run in a single column mode on one processor and to also run high resolution, fully coupled model setups that can run on thousands of nodes. The vast user community also requires CESM

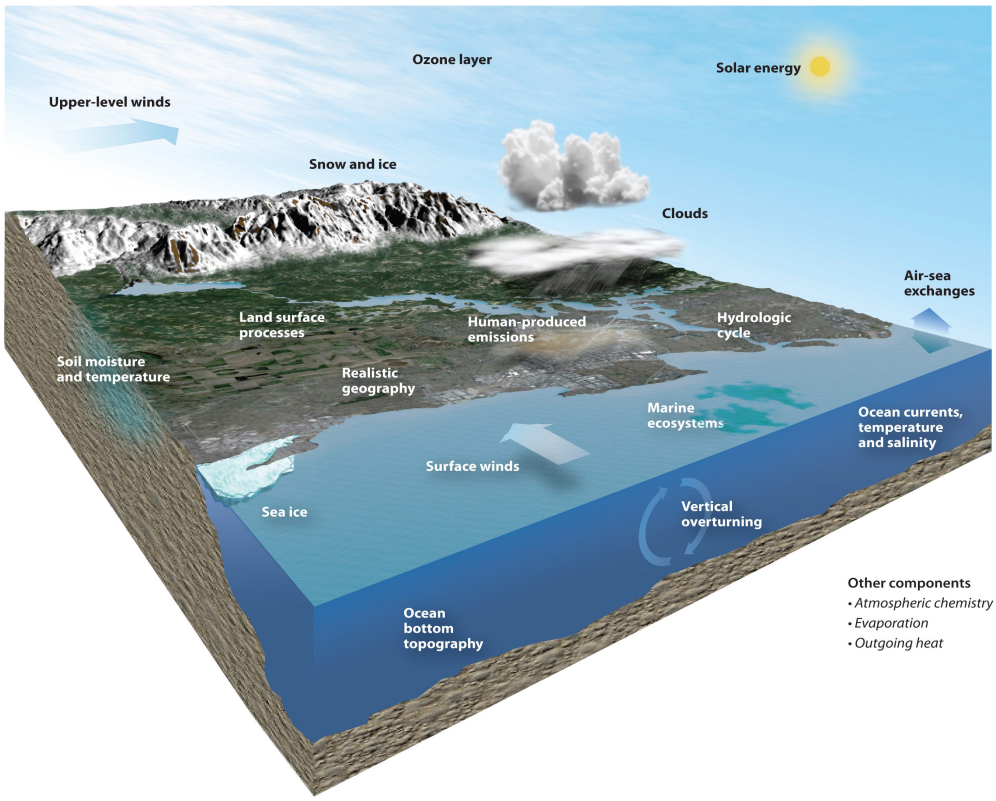


Figure 2.1: The complexity of climate modeling and the processes that are modeled. Image credit: https://www2.ucar.edu/news/backgrounders/complexity-climate-modeling.

to support a variety of compilers that the community uses. Currently, the model supports builds using the Intel, PGI, GNU, NAG, and XL compilers.

CESM uses the Community Atmosphere Model (CAM) [12] as its atmosphere model. The CAM model is used to simulate several physical properties. These include moist turbulence, shallow and deep convection, precipitation, cloud microphysics and macrophysics, aerosols, vertical phases, radiative transfer, and surface exchanges [12].

At experiment setup time, CAM allows users to select additional package options in addition to its default configuration. One option is to run with an atmospheric chemistry model which solves and tracks different chemical species and the reactions that happen within the atmosphere [7]. Adding this option slows down the overall model performance as it traces each of the species through its lifetime.

CAM can also be configured with the Whole Atmosphere Community Climate Model (WACCM). WACCM provides CAM the ability to model up to the lower thermosphere, or up to 140 km [7]. The increased number of vertical layers within WACCM reduces the performance of the model by a factor of five.

CAM consists of two different computational phases, dynamics and physics. The dynamics updates the flow of the atmosphere while the physics computes precipitation, clouds, radiation, and mixing [13]. These phases are currently being computed sequentially in order to ensure numerical stability, but there are ongoing efforts to compute these phases in parallel.

The physics calculations are decomposed in the horizontal direction, with a set number of vertical columns given to each MPI rank. The set number is based on the total number of cores given to the atmosphere by the user at run time. OpenMP is then used to parallelize the computations within the column [13].

The dynamic calculations are computed on either a finite volume or spectral element grid. Both of these grids decompose the domain in the horizontal direction and assign MPI ranks to blocks, but they differ in how they parallelize using OpenMP. The finite volume grid is a latitude, longitude, and level grid that converges at the poles. For this grid, OpenMP is used at a loop-parallel level. The spectral element grid is a cubed-sphere grid where all grid spaces are homogeneous. The elements are arranged in a two-dimensional data structure with horizontal element and level being the two dimensions. For this grid OpenMP is used to parallelize over the vertical level [13].

The physics and dynamic phases are computed on separate grids. Ideally, when each phase is decomposed, the grid cell assignments are aligned closely between the phases in order to reduce the communication time, but this is not always the case and it has an impact on the performance of the model [13]. This poses a problem particularly for the physics model because it is prone to load imbalance due to geographic location and time of day. This presents a challenge to decompose the physics grid to be load balanced as well as aligned with the dynamics grid. This problem has existed within the atmosphere model for a long time [14] and has been a difficult problem to address.

The land model within CESM is the Community Land Model (CLM) [15]. CLM is a single column model that models the terrestrial ecosystem, which includes transfers of energy, water, and ecosystems, the carbon-nitrogen cycle, urban areas, irrigation, and dynamic land uses [7].

CLM is decomposed over the horizontal direction and MPI ranks are assigned a block of grid cells. Each grid cell is composed of multiple landunits. Landunits are composed of snow and soil columns which also contain multiple plant functional types. OpenMP is used to parallelize over these smaller elements within the blocks [13].

CLM uses MOSART for its river runoff model. MOSART is used to model river water transport of freshwater to the oceans [16]. It is a smaller component and has a small impact on performance.

The ocean model within CESM is the Parallel Ocean Program version 2 (POP2) [17]. POP2 models the ocean dynamics by using a set of primitive equations which solve tracer transports and momentum. It also computes temperature, salinity, and age throughout all levels.

POP consists of two computation phases, the baroclinic and barotropic phases. The baroclinic phase is more computational bound and is embarrassingly parallel. The barotropic phase is communication bound. This is because it uses an iterative solver that requires a large amount of high latency communication [13].

The three dimensional ocean grid is decomposed in the horizontal direction, with each MPI rank getting a group of blocks to operate on. OpenMP parallelization is then used over the blocks within each group given to the rank.

There are two different decomposition strategies used with POP, Cartesian and Spacecurve. The Cartesian method works well for smaller core counts. This is because it results in a large amount of halo updates, especially at larger core counts. For medium to large core counts, the Spacecurve method is preferred. The Spacecurve method first eliminates all of the land points. It then calculates a specific ordering of the remaining blocks and then they are assigned to MPI ranks. This gives the Spacecurve method an advantage of balancing computational load better than the Cartesian method [13].

Both POP2 and CLM have the ability to model biogeochemistry [7]. When used within POP2, it is able to model nutrient distributions, the plankton functional types, and the carbonate chemistry of the ocean. When used within CLM, it is able to model the full soil chemistry including both the carbon and nitrogen cycles by simulating photosynthesis, the plant and soil respiration, and litter.

The POP model uses the WAVEWATCH III model to model the Langmuir mixing process in the ocean [18]. This helps improve the mixing of the shallower levels of the ocean. This particular modeling component has been shown to have poor scaling, but it has little effect on the overall performance of CESM if it is run concurrently with CLM and CICE models. This is because it typically takes less time than the CLM and CICE models to run.

The CESM model uses the Community Ice Code (CICE) [19] model to model sea ice. The CICE model calculates the thickness and the extent of sea ice based on the external forcings it receives [7].

The CICE calculations are usually calculated on the same grid as the ocean grid, with the exception that the vertical coordinate represents the thickness of the ice. The grid is decomposed in the horizontal dimension into different blocks, of which groups of blocks are assigned to MPI ranks [13]. The CICE model has several different decomposition strategies and choosing the correct strategy is essential because the CICE model is prone to load imbalance. The load imbalance is a result from almost all calculations being computed only in places on the Earth where there is sea ice and most of the heavy computation is done only where the sun is shining. Therefore it is important to choose the correct strategy that load balances the calculations and reduces the amount of communication for the user selected core count. The different strategies and their strengths and weaknesses are documented in [20].

The land ice model used within CESM in the Community Ice Sheet Model (CISM). CISM models the energy, mass, and momentum of ice sheets on Greenland. It also simulates the flow of the ice sheet and the calving of icebergs [7]. This model is coupled to the land model and is setup through configurations at compile time. CISM is not a computationally heavy model and does not have a large effect on the performance of CESM.

CESM has the ability to output the current state of the model through time slice output files. It also has the ability to write checkpoint restart files. Both output and checkpoint files are written at frequencies specified by the user. Output files are usually written at a monthly frequency, but depending on the type of experiment ran, users may want daily, hourly, and sub-hourly output files. Restart files are usually outputted once every 6-12 compute wallclock hours.

CESM uses the Parallel I/O (PIO) [21] library to output files. PIO provides an interface for users to write data in the NetCDF format using different libraries with one interface. The objective of PIO is to provide better write performance. The PIO library uses the same compute nodes to write as it does to run the model. Therefore, the CESM model component must pause each time output is written. This is a problem when writing high temporal and spatial data as it can take just as long to write output as it does to run the model.

2.2 Model Coupling

Within CESM, all of the modeling components interact with each other through a coupling interface, CPL7 [22] [23]. The CPL7 contains a top level driver which controls the sequencing, time stepping, and communication between the components.

As the CESM simulation timesteps, the individual components receive data from the coupler, run, and then send data to the coupler. This process repeats until the simulation is complete. The general concept is depicted in Figure 2.2 and shows the hub and spoke communication pattern between the coupler and the components. This synchronization is controlled by the CPL7's driver. The parallelism between all of the components is determined by the discretization of the individual components across the available cores, the physical constraints between the components, and the dependencies between certain components.

While the CPL7 driver controls the synchronization, communication, and time stepping of the modeling components, it uses the Model Coupling Toolkit (MCT) [24] to translate the data between the components. One common problem of multiphysics models is that the computations are performed on different grids and different processor sets. Before data is exchanged between

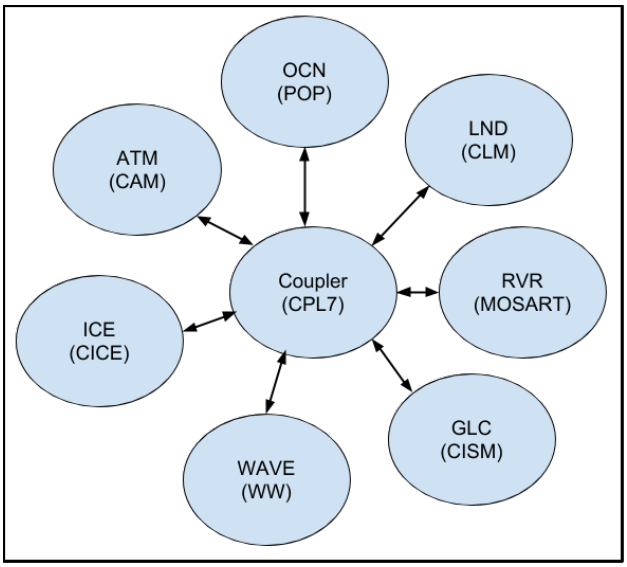


Figure 2.2: The hub and spoke communication between the coupler and all of the modeling components.

different components, MCT must interpolate the data values to the destination grid and correct processor mappings must be created in order to send data to the correct ranks of the destination component.

2.3 Future Model Plans

The model developers of CESM are currently working towards version 3 of the model. In this version, they are expected to replace the POP ocean model with the Modular Ocean Model (MOM) version 6.0 [25]. MOM was developed at the Geophysical Fluid Dynamics Laboratory (GFDL) at the National Oceanic and Atmospheric Administration (NOAA). The work of incorporating MOM into the coupling infrastructure is currently underway [26].

Another large change planned for version 3 of CESM is replacing MCT with the Earth System Modeling Framework (ESMF) [27] as its primary coupling software. While ESMF has been an option within CESM for several years, more emphasis is being put into its development and it will become the default coupling infrastructure within CESM for this upcoming version.

Even though the work in this paper used MCT, the results reported should be similar to what will be seen with ESMF. The coupling idle times will still be a performance bottleneck with ESMF

and the load balancing techniques used in this work will be able to be used with the new ESMF coupling infrastructure.

Chapter 3

Coupling Between the Model Components

3.1 Coupling Interface

The coupler is tasked with providing a means for component sequencing, communication between components, and providing the coupling interface. This includes interpolating fields between different model grids, rearranging the data to map to different core counts, calculating the fluxes that are passed between the models, and creating diagnostic fields [22].

As stated above, as the CESM simulation timesteps, the individual components receive data from the coupler, run, and then send data to the coupler. The default frequencies in which the components communicate this information to the coupler for a one degree simulation is show within Table 3.1. When the component model simulates that much time, the model then pauses and passes certain fields to the coupler. It does not continue running until it receives data from specified components via the coupler.

These timings change depending on the resolution the model is run at. Higher resolution simulations couple more frequently in order to help resolve the physics correctly.

Table 3.1: The coupling frequencies of the CESM components.

Component	Frequency in model simulated time
Atmosphere	30 min
Land	30 min
Sea Ice	30 min
Ocean	1 hour
Glacier	1 year
River Runoff	3 hour
Wave	30 min

3.2 Sequencing of the Components

The execution sequence of the different components is constrained by where the components are layed out across the available cores and by scientific requirements. The largest scientific constraint that prevents all of the components from running concurrently is that the atmosphere has to run sequentially from the land, sea ice, and river models. This is done in order to better resolve the diurnal cycle [23]. Particularly, this is done in order to pass surface albedo, atmospheric radiation, and boundary layer stability at the correct times in between these models to ensure numeric stability within the simulation [22].

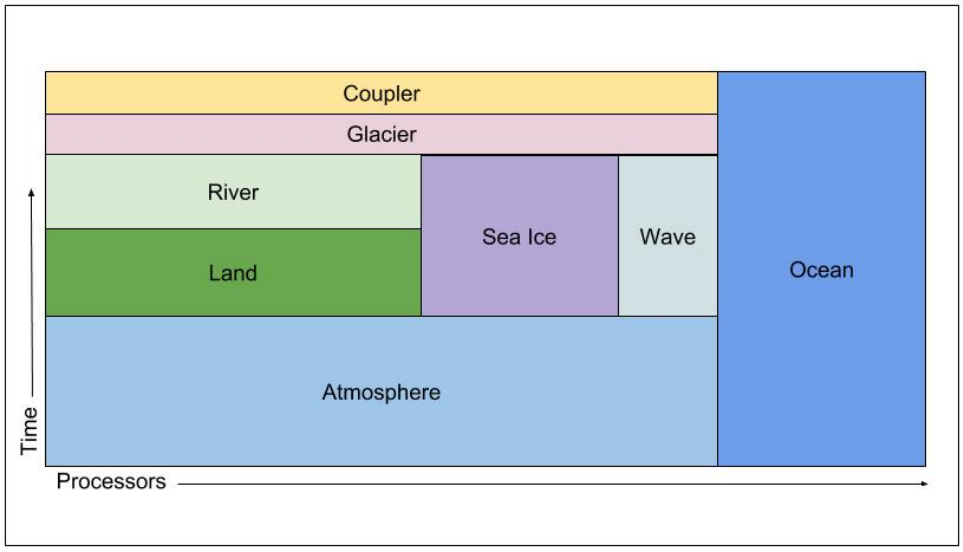


Figure 3.1: The recommended CESM component layout across processors.

Because of these requirements, Figure 3.1 shows the optimal layout that the model components should placed on the available MPI ranks. At this particular layout, all of the components that can be run in parallel do not share any MPI ranks.

As CESM steps through time, each of the components are run in a set sequence. This sequence is shown in Figure 3.2. This sequence is hard coded in the model and cannot be changed through user configurations.

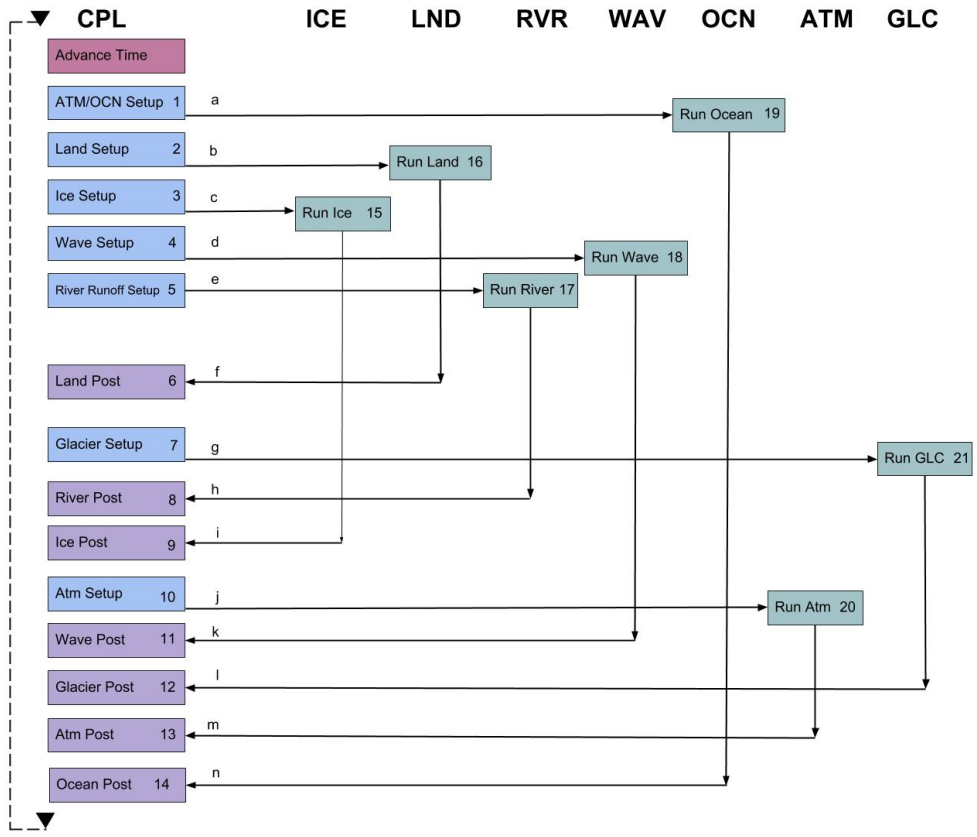


Figure 3.2: The calls within the cimecompmod.F90 code that communicate to and from components and the coupler and the running of each of the components. Boxes indicate tasks, while arrows indicate communication. The blue color represents the coupler sending information to a model component. The purple color represents a component sending data to the coupler. Aqua represents a running component. The number and letter labels should be used to identify the timing results in Table B.1 and Figure B.1.

The sequencing shows that a given model must first receive specified fields from the coupler. After this, it runs until a specified amount of time has been simulated. It then passes specified fields back to the coupler. The coupler then passes these fields to other components that are waiting for them. The original component continues to pause until it receives all of the fields it needs before it can continue on. This pattern continues until the simulation completes.

3.3 Communication Pattern

In order to capture which component MPI ranks communicate with which coupler MPI ranks, a test simulation was run where all components were run on unique MPI ranks and communication patterns were outputted and mapped. This simulation was run at the lowest production layout, which is a two degree atmosphere grid, with a one degree ocean grid, in order to make the mappings easier to manage because of the lower MPI rank counts. Table 3.2 shows which set of unique ranks each of components was run across.

Table 3.2: The MPI ranks used in Figure 3.3 and Figures A.1–A.8.

Component	Range of MPI Ranks
Coupler	0–179
Atmosphere (CAM)	180–359
Land (CLM)	360–434
Ice (CICE)	435–539
Ocean (POP)	540–659
River Runoff (ROF)	660–734
Glacier Model(CISM)	735–749
Wave (WAV)	750–879

As seen within Figure 3.3, each of the components have unique communication patterns with the coupler. Some components communicate with all coupler ranks and their own ranks while others communicate with less ranks. Each pattern is influenced by how the data is decomposed on the model component's native grid.

These patterns can also be seen in the network maps found in the Appendix A.1 though A.8. In each map, for a given rank assigned to a particular component, you can see which nodes within the coupler it communicates with. In some cases, the component must communicate with other ranks assigned to this component in order to collect all data that is needed to send to the coupler.

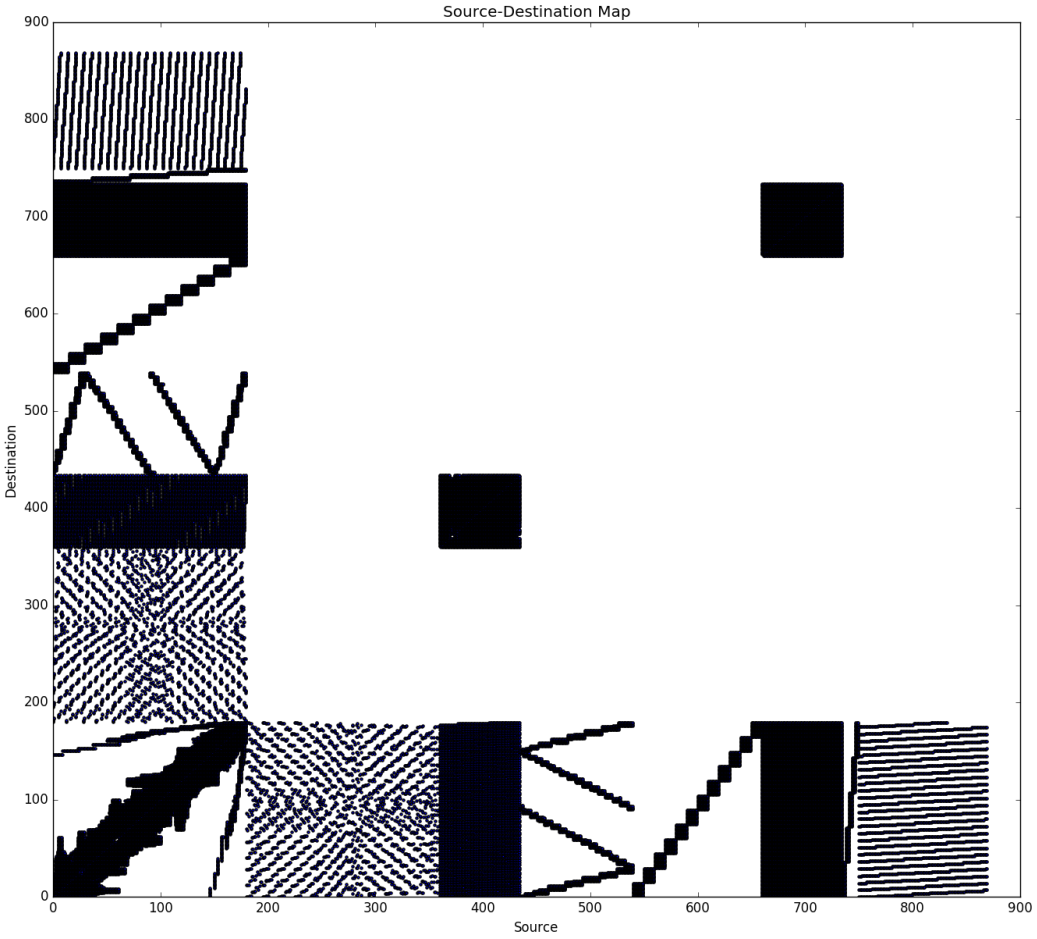


Figure 3.3: Destination ranks and source ranks communicated to and from the coupler.

Chapter 4

Computing Platforms and Problem Size

4.1 Computing Platforms

4.1.1 Architectures

The work described in this research was performed on two separate machines. This was because the first machine was retired and replaced by the second while this research was being conducted.

The first architecture was yellowstone [28] [29], a 1.5 petaflops IBM iDataPlex cluster. It consisted of 4,662 nodes, each with a dual socket Intel Sandy Bridge EP connected by a Mellanox FDR InfiniBand full fat-tree network. The system was connected to a GPFS storage system. Yellowstone was used in production from 2012 until the end of 2017.

The second architecture used for this research was cheyenne [30], a 5.35 petaflops SGI ICE XA cluster. It consists of 4,032 nodes, each with a dual socket Intel Broadwell connected by a Mellanox EDR Infiniband 9-D enhanced hypercube topology network. The system is connected to the same GPFS storage system that yellowstone was connected to. The system has been in production since the beginning of 2017 and had a year overlap with the yellowstone system.

4.2 Problem Sizes

4.2.1 Model Configuration

The model configuration is determined by the type of science to be explored. CESM is designed to be flexible enough to allow data models to be substituted in the place of any of the component models. In this case, prescribed values are read from an input file at coupling time, instead of passing computed values. In some setups, only one prognostic component model is used with the remaining being data models. In the other extreme, all of the prognostic components models can

be run. Stub models are also used, but these do not provide data to the system. Instead, they are only there to satisfy an interface requirement.

In addition to considering which components are ran, different components have different science options that affect the performance. The different science options use different calculations, that can be used as an addition to or replace standard calculations. Other options add grid cells in the vertical direction and add extra calculations.

This presents a very large combination of different ways the model performs. CESM uses specific component set names to in order to replicate specific model combinations and science options. Specific component model and data model combinations are identified by the first letter in the name and the remaining characters in the name specify the science objective. Some of the commonly used first letters for the model combinations are shown in Table 4.1.

Table 4.1: Some of the common combinations of prognostic and data models that can be run in CESM. If a model type is not listed, a data or stub model is used instead.

First Letter	Category
A	All data models
B	All active components
C	Active ocean model
D	Active ice model
E	Active atmosphere and ice models
F	Active atmosphere and land models
G	Active ocean and ice models
I	Active land model

The performance testing presented here used the CESM model combination name B1850. Because it starts with the letter B, this indicates that it uses all of the model components and no data models. This simulation is a hindcast that simulates a pre-industrial climate, holding the climate at 1850 conditions. It uses the standard number of model levels and uses the out-of-the-box component configurations.

4.2.2 Grid Sizes

The different CESM grids being run in production are shown in the Tables 4.2 and 4.3. In both, tables a degree refers to a degree in longitudinal and latitudinal space and the spatial distance is dependent on the type of grid and a grid point's location on the Earth.

(a) The Spectral Element Grid **(b)** The Dipole Grid **(c)** The Tripole Grid

Figure 4.1: Some of the different grids used in CESM. Grid (a) is the Spectral Element Grid and is one of the grids used by the atmosphere and land models. Grids (b) and (c) are the Dipole grid and the Tripole grid, respectively, and both are used by the ocean and sea ice models. Image credit: https://www.earthsystemcog.org/projects/dcmip-2012/cam-se and http://www.cesm.ucar.edu/.

Some of these grids are shown in Figure 4.1. Grid (a) is the spectral element grid used by the atmosphere and land models. This is a cubed sphere grid where all points are homogeneous. Grid (b) is the dipole grid used by the ocean and sea ice models. This grid relocates the poles to be over Greenland and Antarctica to avoid performing calculations at the convergence points. Grid (c) is the tripole grid used by the high resolution ocean and sea ice models. This grid relocates the poles to be over Russia, Alaska, and Antarctica. As with the dipole grid, this is done to avoid performing calculations of the convergence points.

The most popular grids being used in production are the one degree finite volume grid, f09, for the atmosphere and land models and the one degree dipole grid, g17, for the ocean and ice models. These grids are more commonly run because the throughput of the model is sufficient and it does not sacrifice some science that is not resolved on the courser grid. The quarter degree spectral

element grid, ne120, and the tenth degree tripole grid, t12, are considered high resolution are run sparingly depending on allocations awarded because it is considerably more expensive to run.

Table 4.2: The different grid sizes used by both the atmosphere and land models. FV refers to the finite volume grid and SE refers to the spectral element grid.

Short Name	Nickname	Grid Type	# of Lat Points	# of Lon Points	# of Horiz Points
f19	2 degree FV	Finite Volume	96	144	
f09	1 degree FV	Finite Volume	192	288	
ne30	1 degree SE	Spectral Element			48602
ne120	1/4 degree SE	Spectral Element			777602

Table 4.3: The different grid sizes used by both the ocean and sea ice models.

Short Name	Nickname	Grid Type	# of Lat Points	# of Lon Points
g17	1 degree	Irregular Dipole	320	384
t12	1/10 degree	Irregular Tripole	3600	2400

Unless otherwise noted, the performance work done in this document was simulated with the one degree finite volume and one degree dipole grid. This was chosen because it is the most commonly used grid in production runs and, thus it was important to study and improve the performance of it in order to make the largest impact.

4.2.3 Output

The amount of output generated by the model is configurable by the user. By default, each modeling component outputs a file for each month that it simulates. The files contain several output climate variables that each consist of a single floating point value for each grid point in the vertical and horizontal directions, for each time.

The user has the option to vary the number of variables outputted by each component model and well as the time frequency that is outputted. By default, the output is outputted at the end of every month simulated, but users have the additional option to output variables on yearly, daily, hourly, and sub-hourly frequencies.

Outputting data on a monthly basis has a small impact on performance, but higher frequency data can have a high negative impact on performance. The time it takes to output data is recorded in the overall component run time because PIO uses the compnenet's ranks to write out its data. This causes the simulation to pause every time data is written to disk. Some output configurations have the possibility to double the time it takes for a modeling component to run.

The performance results within this work have all output turned off in order to capture true model performance. In reality, the time it takes for a given model component to run should be higher depending on the amount of output the user specifies.

Chapter 5

Performance Results

5.1 CESM Timers

When evaluating the performance of CESM, a first step is to evaluate the timing numbers from the model. With each CESM simulation, a set of default timings are outputted within two timing output files. The first file is a summary file with higher level timers. The second file contains timers that are found deeper within the code and provide a higher resolution of timing results. In order to get more timers, CESM provides a timer depth variable that allows users to output more or less timers based on the function call depth they are interested in.

The timings numbers shown in Table B.1 have been gathered from two separate CESM simulations at different resolutions in order to show the performance impact of varying resolutions. The 1 degree atmosphere, 1 degree ocean results were from a fully coupled run that ran with all of the component models. The 1/4 degree atmosphere, 1 degree ocean results were also from a fully coupled simulation. The higher resolution results were run with CESM version 1.3, which was the previous release of the current release. The model run did not contain the wave model nor the glacier model, and thus these numbers are missing in the Table.

In order to interpret the timing results better, Figure 3.2 contains labels have been added to each line. The labels match the Label column in Table B.1.

When comparing the coupler timing results (labels 1-14) against the component run times (labels 15-21) in Table B.1, you can see that only a small fraction of the total run time is actually spent within the coupler interpolating values and calculating the fluxes. Where we see more variance within the results is within the labels a-n. These timing results have a portion of the barrier wait times within them and you can see that some of the timings can be quite large compared to the total run times of the components.

The largest differences between the 1 degree and 1/4 degree simulations are seen in the atmosphere and ocean communication back to the coupler. A closer evaluation of the timing numbers showed that a majority of the difference in the timing numbers were spent in barriers. This is a result of the components being less balanced in the higher resolution experiment than in the lower resolution experiment. These timing numbers can be improved by modifying the number of MPI ranks the components are run on in order to load balance the run.

5.2 IPM Results

In order to better understand the coupler's communication performance, Integrated Performance Monitoring (IPM) [31] was used to profile a one degree atmosphere, one degree ocean model simulation. Only the lower resolution model was profiled because the overhead of IPM is very high. The cost of IPM is correlated with the number of MPI ranks it has to collect data on, which made collection on the highest resolution very difficult. While the profiling only occurred at a lower resolution, anything discovered in coupler performance found at this resolution can be transfered to the higher resolution.

Figure B.1 shows the proportion of time spent in each of the communication steps within the coupler. In this figure, each of the pie charts are labeled to match the labels within Figure 3.2. In each of the steps, a majority of the time was spent in synchronization calls because of load imbalance between the different model components. This happens when the components are not given adequate resources in order to synchronize at similar times. Instead, components are left idling, waiting for other components to catch up.

The performance of this particular simulation could be improved by properly load balancing the component models. This is done by balancing the resources between the components based on their overall run times. This requires expert user knowledge and an understanding of the scalability of each of the components.

5.3 Coupler Performance

The performance of the coupler is bounded by the communication barriers and the computational time to regrid and map the values sent between the modeling components. More often than not, the time spent in the coupler is dominated by the communication barriers, as indicated by the two performance capturing methods described above. The communication barriers where the most time is spent are the barrier calls in between different component calls. At a given time, a component pauses and has to wait for another component to get to the same timestep. At this time, values are passed between the models and each model can then continue. Ideally this process would happen close in time, but it is difficult to achieve a perfect load balance between all components within CESM.

Chapter 6

Automated Load Balancing

As noted in Chapter 5, a majority of the time spent in the coupler was spent in synchronization calls. This happens when modeling components have to wait for other modeling components to catch up. Therefore, it is important to allocate the correct number of resources to each component in order to have each component call the coupling interface at the correct times.

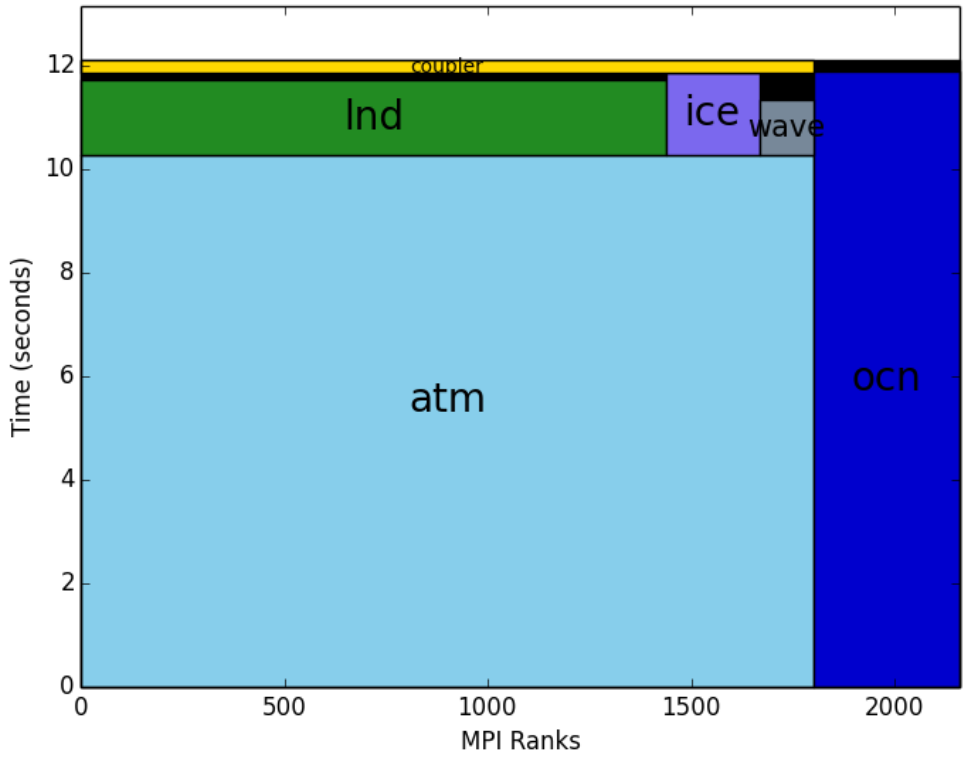


Figure 6.1: CESM component layout across MPI ranks. The size of the boxes indicates to what proportion of the total number of MPI ranks that should be given to each component for a one degree atmosphere, one degree ocean fully coupled simulation. The black sections show the idle time spent waiting for another component.

Figure 6.1 is shown to demonstrate how to align the modeling components in order to load balance the system. The figure shows a layout that is very close to being load balanced because

the black areas are relatively small. It is very difficult to have perfect load balance between the components because of the nonlinear scaling of some of the models and node alignment.

$$(ATM + max((LND + ROF), ICE, WAV)) == OCN \tag{6.1}$$

Load balancing the fully coupled CESM involves following the rules defined in Equation 6.1 using the above model alignment in Figure 6.1. In order to load balance a fully coupled CESM experiment, the ocean model must couple at the same time as the atmosphere model plus the maximum component run time between the land plus river, or the ice, or the wave model.

Along with the timing numbers that report specific times for subroutines, full component run times are also reported within the timing files. The timing results are outputted into a timing directory that is placed in the CESM experiment case directory and is named cesm_timing*. A sample is shown below:

```
Runs Time in total seconds, seconds/model-day, and model-years/wall-day
CPL Run Time represents time in CPL pes alone, not including time associated
with data exchange with other components

TOT Run Time: 263.167 seconds   13.158 seconds/mday      17.99 myears/wday
CPL Run Time: 4.733 seconds     0.237 seconds/mday     1000.26 myears/wday
ATM Run Time: 205.649 seconds   10.282 seconds/mday      23.02 myears/wday
LND Run Time: 28.776 seconds    1.439 seconds/mday      164.52 myears/wday
ICE Run Time: 31.759 seconds    1.588 seconds/mday      149.07 myears/wday
OCN Run Time: 38.042 seconds    11.902 seconds/mday      19.89 myears/wday
ROF Run Time: 0.401 seconds     0.020 seconds/mday    11806.10 myears/wday
GLC Run Time: 0.008 seconds     0.000 seconds/mday   591780.82 myears/wday
WAV Run Time: 21.169 seconds    1.058 seconds/mday      223.64 myears/wday
ESP Run Time: 0.000 seconds     0.000 seconds/mday        0.00 myears/wday
CPL COMM Time: 2.657 seconds    0.133 seconds/mday     1781.80 myears/wday
```

To load balance the model, it is best to evaluate the timing numbers in the first column. These timings numbers are the total seconds it took to run each of the different components. By in-

putting them into Equation 6.1, users can experiment with different core counts in order to find a combination that makes the equation true.

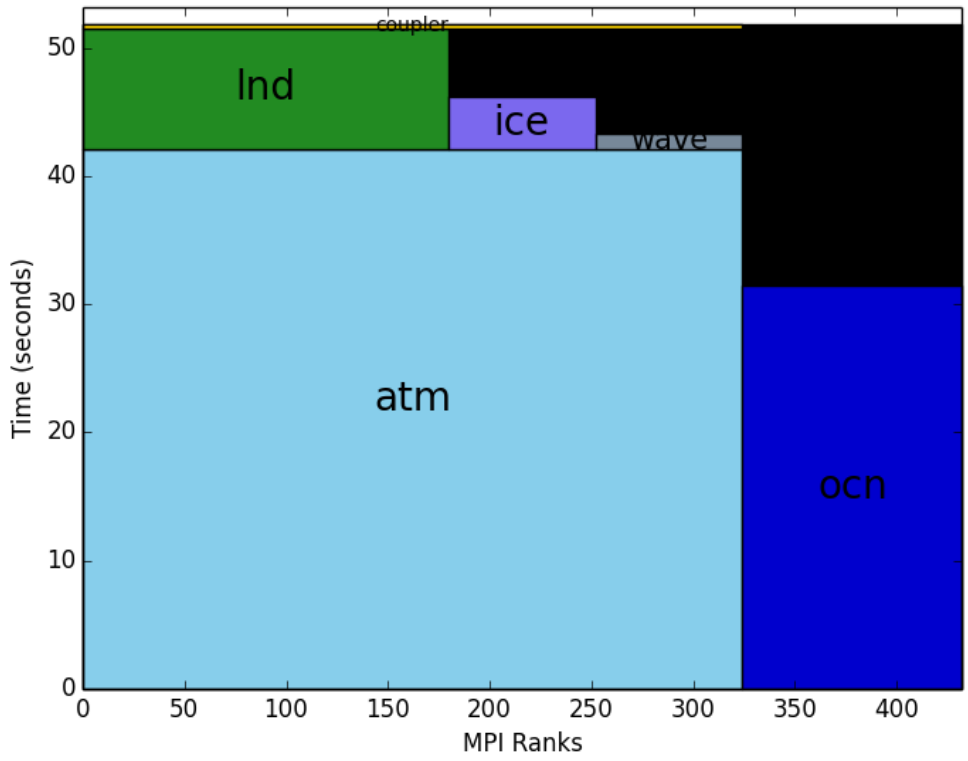


Figure 6.2: An unbalanced CESM layout. The size of the boxes indicates to what proportion of the total number of MPI ranks that should be given to each component for a one degree atmosphere, one degree ocean fully coupled simulation. The black sections show the idle time spent waiting for another component.

As an example, an unbalanced layout is shown in Figure 6.2. As you can see, there is a lot of black which indicates idle time where the components are waiting for other components to catch up. In this example, too many resources where given to the ocean model and it is completing too quickly. This is also the case for the ice and wave models compared to the land model. In order to load balance this case, more resources need to be given from the ocean model to the atmosphere model. And in order to load balance the land and ice components, some of the ice resources need to be shifted to the land model.

Load balancing CESM requires expert knowledge of the coupling infrastructure as well as the scaling performance of all of the components. A typical scientist running the model does not have this knowledge and they must rely on a software engineer to provide them with a component layout before any experiment is run. As noted earlier, this is a common problem with all coupled climate models [10]. Creating a tool that automates this process would allow the scientists to discover optimal component layouts for their experiment, potentially saving hundreds of thousands of core hours per experiment. This would also have the potential to save many core hours around the world because similar component layouts are used in other coupled climate models as well [10].

6.1 Load Balancing Tool using MINLP formulation

At the beginning of 2013, an automated load balancing tool was created by Alexeev, et al in [1] [32]. In this work, we developed a two stage static load balancing process to help find optimal component layouts.

The first step in the process collected benchmark results through running the model at different core counts. While doing this work, we stated that this was the weakest part of the process because it relied on the user to select core counts that accurately depict the full scaling curve of the modeling components. It was also the most time consuming process because it requires several model runs to collect this data. In the paper we suggested running, at a minimum, four model runs. This included one with the fewest number of cores a component can run on, a second with the most number of cores a component can run on, and two other counts that reside in the middle of this range.

The second step involved solving for the optimal component layout across a set total number of cores. In order to solve for the complex relationship between the number of cores and the run times, a mixed-integer nonlinear optimization (MINLP) problem was formulated. In order to solve the MINLP problem, both a mathematic model of the CESM component layout constraints and a performance model must be formulated that describe the problem.

The MINLP problem was written in AMPL [33] and was run using the toolkit MINOTAUR [34]. The particular MINOTAUR solver we used was the LP/NLP branch-and-bound solver [35] which performs a search, partitioning and sampling the search space, finding the optimal solution.

The results from the first step were used within the mathematical model and solved for a target task count. The results included both the recommended core counts for each component and the estimated run times for each component.

The variables used in the objective function are defined in Table 6.1. The objective function used to solve the MILP are shown in Equation 6.3. This function contains three pieces that represent the scalable, serial, and missing time contributions and looks to minimize Equation 6.4. This function is subject to the constraints listed within Table 6.2. These are the same constraints that are depicted in Figure 6.1. All of these variables, constrains, and functions are found within [1].

$$max(max(ice, lnd) + atm, ocn) \tag{6.2}$$

When the paper by Alexeev, et al. [1] was written, an additional constraint was needed to account for the ocean model needing to be run at specific core counts. This was because the decomposition strategies were hard coded into the model and users were required to run the ocean model at these specific core counts or it would create a compile time error.

$$T_j(n_j) = T_j^{sca}(n_j) + T_j^{nln}(n_j) + T_j^{ser} = \frac{a_j}{n_j} + b_j n_j^{cj} + d_j \tag{6.3}$$

$$min(a_\mathrm{j}, b_\mathrm{j}, c_\mathrm{j}, d_\mathrm{j}) \sum_{j=0}^{D_j} (y_{ji} - \frac{a_j}{n_{ji}} - b_j n_{ji}^{cj} - d_j)^2 \tag{6.4}$$

Table 6.1: The variables used within Table 6.2 and Equations 6.4 and 6.3. All of the variables must be positive integers or positive real numbers. These were taken from [1].

Variable	Description
i	component core count
j	component model
C {i, l, a, o}	set of components {ice, lnd, atm, ocn}
T	total wall clock time to run the model
N	total number of cored
T_{icelnd}	wall clock time to balance the lnd and ice models
T_{sync}	synchronization tolerance to balance the lnd and ice models
n_j	number of cores to allocated per model
$T_j(n_j)$	performance function that models the time to run a component on n_j
$z_k\{0,1\}$	binary variables used to model the selection of the number of cores
T_j^{sca}	time spent in scalable code
T_j^{ser}	time spent in serial code
T_j^{nln}	time spent in initialization, communication, synchronization code
D_j	total number of data points available for the fitting function
a_j,b_j,c_j,d_j	fitting parameters used in the performance function for each model
n_{ji}	number of cores allocated to that model
y_{ji}	time taken to run a component with n_{ji} cores

Table 6.2: Constraints used for the the MINLP. These were taken from [1].

Subject to these constraints:	
T_{icelnd}	>= $T_i(n_i)$
T_{icelnd}	>= $T_l(n_l)$
T	>= T_{icelnd}+$T_a(n_a)$
T	>= $T_o(n_o)$
$T_l(n_l)$	>= $T_i(n_i)$-T_{sync}
$T_l(n_l)$	<= $T_i(n_i)$+T_{sync}
n_a+n_o	<= N
n_i+n_l	<= n_a

The parameters a_j,b_j,c_j,d_j used within the objective function shown in Equation 6.3 are estimated by evaluating the benchmarking data that contained the time to run each component model at the different node counts. As we note in the paper [1], there are two main factors that effect the quality of these estimated parameters. First, the estimated parameters are heavily influenced by the benchmarking core counts used. If the benchmark fails to fully capture the characteristics of the performance curve, the estimates will not be accurate. Second, the benchmarking results create a rough scalability curve that does not account for load imbalances within the modeling components.

This type of imbalance occurs when the individual model component's decomposition is poorly auto-selected. While the cost of the imbalance is represented within the cost of the component model that is fed into the objective function, the total cost of the imbalance is not always linear and thus may create inaccuracies within the predicted run times.

The original version of the load balancing tool did a good job predicting the run times per component and was able to find better component layouts than were being used at that time. It did have problems predicting the performance of the sea ice model, though. This is because the performance of the sea ice model is not linear and is highly susceptible to intra-component load imbalance. This behavior in the CICE model is documented in [36], [20], and [37].

6.2 Load Balancing Tool using MILP (linear) formulation

The usage of MINOTAUR in the original version introduced the dependency of the load balancing tool having to run on a special server located at Argonne National Laboratory and this limited the usablility and lifetime of the tool. Therefore, a new version of the tool [2] was written by Jason Sarich in Python to help with portability and to fit within the CESM infrastructure.

As with the original version of the load balancing tool, the newer version of the tool is also a two step process. The user is still required to run a series of benchmarks to collect performance results from all of the components. The second step again collects the results, processes them, and feeds them into a solver. This new version uses the COIN-CBC solver within the library PuLP [38] to solve for the mixed integer linear optimization problem (MILP).

The newer version of the code uses a linear solver, unlike the original which used a nonlinear solver. This has the benefit of being a simpler model which runs faster, but it provides only an approximate solution. This type of model makes an assumption that each component has perfect scalability, which is not the case. To compensate for this, the time it takes to run each component is computed as the sum of a factor determined by the drop in efficiency and the original time. This becomes the new cost of running the model.

The MILP version of the load balancing tool looks to minimize the total cost, or time to run CESM. This is described within Equation 6.5. In order to approximate the total cost for the full model, each component model's cost must also be approximated. This is done by evaluating the slope between two data points, which is calculated from the cost of the model over the number of nodes. In order to complete the range, the tool must extrapolate a cost value for just one task for each of the components. This presents a weak point in the prediction model because the estimation is usually incorrect.

$$max(max(ice, lnd, wav) + atm, ocn) \tag{6.5}$$

The variables used in the objective function are defined in Table 6.3. The objective function used to solve the MILP are shown in equation 6.8. This function is subject to the constraints listed within Table 6.4. These are the same constraints that are depicted in Figure 6.1. All of these variables, constrains, and functions are found within the load balancing tool documentation found at [2].

Table 6.3: The variables used within Table 6.4 and Equation 6.8. These were taken from [2].

Variable	Description
c	component
i	timing data instance
N[c]	number of cores for component c
NB[c]	the requested core count multiple for component c
C[c]_i	time (or cost) to run each component at each timing data i
N[c]_i	number of cores used to run each component at each timing data i
T[c]	total cost for component c
T1	the maximum cost between the lnd, ice, and wav models
T	the maximum cost between T1+atm and the ocn model

Table 6.4: Constraints used for the MILP. These were taken from [2].

Subject to these constraints:	
T[ice]	<= T1
T[lnd]	<= T1
T[wav]	<= T1
T1 + T[atm]	<= T
T[ocn]	<= T
NB[c]	>= 1 for c in [ice, lnd, ocn, wav, atm]
N[ice] + N[lnd]	<= N[atm]
N[atm] + N[ocn]	<= TotalTasks
N[c]	= blocksize * NB[c], for c in [ice, lnd, ocn, wav, atm]

$$T[c] >= C[c]\_\{i\} - N[c]\_\{i\} * (C[c]\_\{i+1\} - C[c]\_\{i\})/(N[c]\_\{i+1\} - N[c]\_\{i\}) + N[c] * (C[c]\_\{i+1\} - C[c]\_\{i\})/(N[c]\_\{i+1\} - N[c]\_\{i\}), for\ i = 0... c\ in [ice, lnd, ocn, wav, atm] \tag{6.6}$$

Once the equation has been fully iterated over all components (c) and all timing data (i), the resulting MILP data is passed to the linear solver. The solver then returns the estimated times to run each of the components as well as the number a tasks to run each component on, equal to or less than the total number of tasks the user specified.

6.2.1 Improved Performance Results Using the Load Balancing Tool

The MILP version of the load balancing tool was tested to see how close it could approximate the solution. As a first step, five CESM simulations were run on varying core counts in order to capture the scaling curves for each of the components. The results are shown in Figure 6.3.

After the benchmarks were collected, the second step was run to solve for the MILP. In Figure 6.4, you can see the predicted timing results versus the actual timing results.

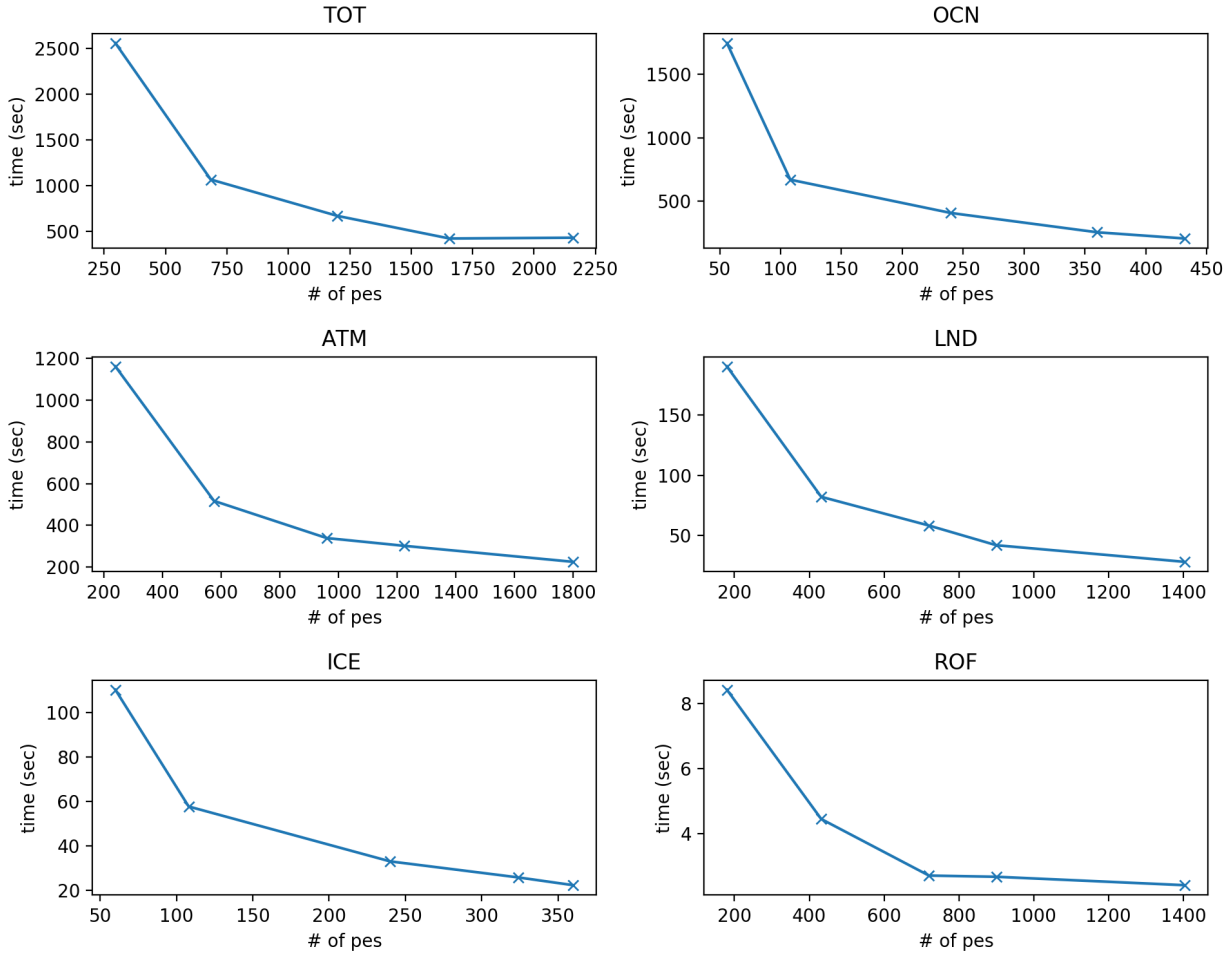


Figure 6.3: The initial scaling results that were used to generate load balanced layouts.

Overall, the model performed well, but it had difficulty predicting the performance of the ice and ocean component models. Performance prediction of the sea ice was also a problem of the original tool. This is due to the intra-component load imbalance within this model caused by poor choices made by the dynamic decomposition strategy that was chosen for those core counts. The MILP version of the load balancing tool is having a problem predicting the performance of the ocean model because it no longer contains the set decompositions that were in the CESM code at the time of the original load balancing tool. The set decompositions allowed the performance of the ocean model to be more predictable. Instead it now uses a similar automated strategy as to what is found within the sea ice model and is now more susceptible to creating load imbalanced decompositions.

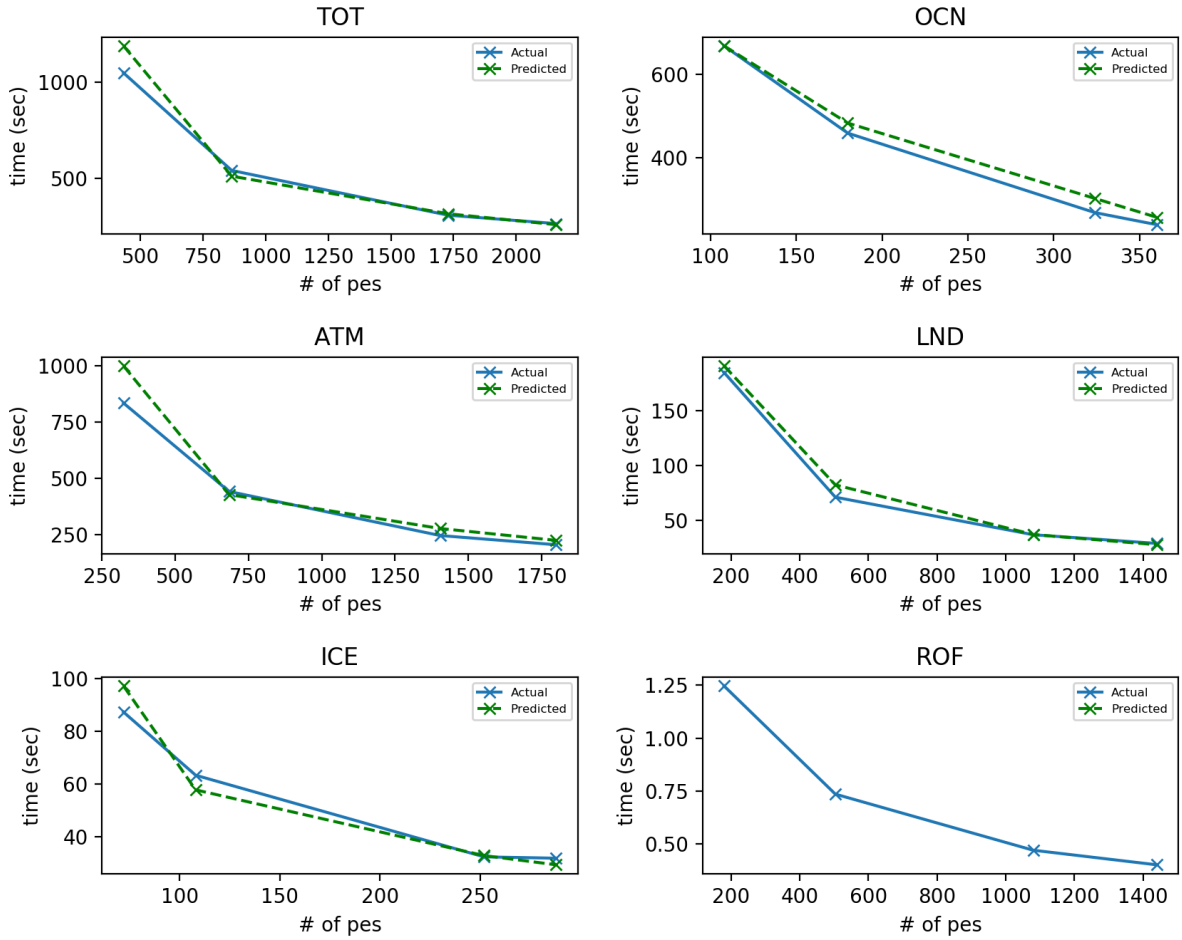


Figure 6.4: The results per component comparing the load balancing tool's predicted performance versus the actual performance.

The difficulty in predicting the performance of the ocean and ice component models has little influence over the overall predicted cost of the model, though. This is because in the tested cases, the ocean model always ran faster than the atmosphere plus the land model and the ice model always ran faster than the land model. Thus neither were the dominate cost and they did not affect the total predicted run time of CESM.

The extrapolated cost value for running each of the model components on one core creates a weak point in the performance model because, in most cases, there are no near data points creating large errors in the resulting value. This contributed to the large error seen approximating the the lowest core counts for the total and atmosphere. Because the atmosphere's benchmarking used

higher core counts, there were no near point to one core, creating a large error in the approximate cost of the model at lower node counts.

Table 6.5: The production layout timing results versus the MILP layout chosen by the load balancing tool. Both sets use 2,160 tasks. The only difference is the proportion of resources given to the different components.

CESM Timing Information	Production Layout	New Layout
Total Time to Simulate One Day (seconds)	13.6	13.2
All Coupler Idle Time for Each Model Day Simulated (seconds)	6.0	5.1
System Idle Time for Each Model Day Simulated (seconds)	1.7	1.5
Simulated Years/Day	17.42	17.99
Total Cost in Core Hours to Simulate One Year	2975.28	2881.68

Despite these problems, the MILP version of the tool was able to find a better load balanced component layout using a total of 2,160 tasks. The better results are shown in Table 6.5. The new layout found by the tool reduced the system idle time by 0.2 seconds per model day simulated and by 0.9 seconds when all coupler idle time from all components are added together. The reduction in idle time allows scientists to create an extra 6 months of simulation per computer wall clock day and saves approximately 93 core hours for every year that is simulated. Since the model is usually used to simulate around 100 years of climate per experiment, this performance improvement has the potential to save 9,300 core hours for every simulation that will be run.

6.2.2 Testing a New Set of Constraints

In order to validate that the current constraints are optimal, we tested a new set of constraints. The new variables are shown in Table 6.6 and are used to solve for the minimal of the maximum end time for each component as shown in Equation 6.7. The new objective function used to solve the MILP is shown in Equation 6.8 and is subject to the constraints shown in Table 6.7.

Table 6.6: The variables used within Table 6.7 and Equations 6.7 and 6.8.

Variable	Description
c	component
i	timing data instance
Ts[c]	component start time
Te[c]	component end time
C[c]_i	cost to run each component at each timing data i
N[c]	number of cores for component c
N[c]_i	number of cores for component c at each timing data i
T	the maximum cost between all models

$$min(max(Te[atm], Te[ocn], Te[lnd], Te[ice], Te[wav])) \tag{6.7}$$

Table 6.7: The new set of constraints tested.

Subject to these constraints:	
Ts[river]	<= Te[land]
Ts[atm]	<= Te[river]
Ts[atm]	<= Te[ice]
Ts[atm]	<= Te[wav]
Te[ocn]	>=Ts[ocn]
Te[atm]	>=Ts[atm]
Te[ice]	>=Ts[ice]
Te[lnd]	>=Ts[lnd]
Te[wav]	>=Ts[wav]
Ts[c]	>=1
Te[c]	>=1
N[c]	= blocksize * NB[c], for c in [ice, lnd, ocn, wav, atm]

$$\begin{aligned} Te[c] >= Ts[c] + (C[c]\_\{i+1\} - C[c]\_\{i\})/(N[c]\_\{i+1\} - N[c]\_\{i\}) + \\ N[c] * (C[c]\_\{i+1\} - C[c]\_\{i\})/(N[c]\_\{i+1\} - N[c]\_\{i\}), \\ for\ i = 0... \\ c\ in [ice, lnd, ocn, wav, atm] \end{aligned} \tag{6.8}$$

In order to make this modification, the wrapper code that generates the constraints that are fed into the MILP were modified. It included modifications to both the constraints and additional variables to include the start and end times for each of the components.

Even though the constraints fed into the model were different, they both produced the same load balanced layouts and the same approximate costs for a 1024 total task count. These results are shown in Table 6.8 and Table 6.9.

Table 6.8: The results of the existing constraints verses the new constraints implemented as part of the validation. The timings represent the time it takes to simulate 10 days of climate. Both produced the same results.

Result Type	Existing Constraints	New Constraints
Cost ATM	446.0	446.0
Cost ICE	62.0	62.0
Cost LND	62.2	62.2
Cost OCN	508.1	508.1
Cost WAV	53.8	53.8
Cost Total	508.2	508.2
NTASKS ATM	849	849
NTASKS ICE	118	118
NTASKS LND	676	676
NTASKS OCN	175	175
NTASKS WAV	55	55
NTASKS TOTAL	1024	1024

Table 6.9: The predicted start and end times for each of the components, as determined by the new set of constraints. The timings represent the time it takes to simulate 10 days of climate.

Component	Start Time	End Time
ATM	0.0	446.0
ICE	466.0	508.1
LND	466.0	508.2
OCN	0.0	508.2
WAV	466.0	499.7

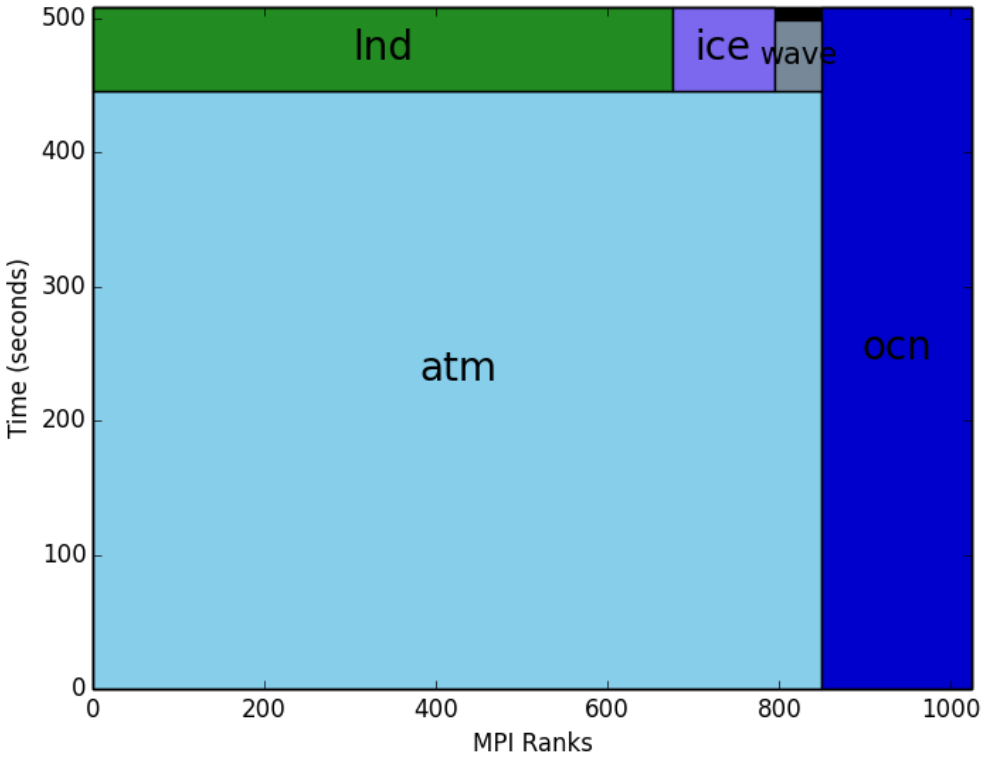


Figure 6.5: CESM component layout across MPI ranks, as determined by the new set of constraints. The timings represent the time it takes to simulate 10 days of climate.

As seen both within Table 6.9 and Figure 6.5, the component end times for the ocean, land, and ice models are all very close in time, with the wave model finishing about 1.5 seconds sooner for every 10 days simulated. If the prediction is fairly accurate, this should produce a load balanced configuration because these results match the results that used the original set of constraints as seen within Table 6.8.

The original constraints are validated by these results, proving that they are able to find the same optimal solution. Proving that the new constraints provide the same optimal solution provides more generality because the particular layout is no longer hard coded into the MILP. This allows it to be more easily adapted to other climate models as well as other types of multi-physics grid problems.

6.2.3 Cost Efficient Solution

The current version of the load balancing tool is able to solve for the best layout given a specific target total task count. This is ideal if you are familiar with the scaling of the application and have a good idea of where your target task count lies on the scaling curve. In order to pick this target task count, a user must consider if it is more important to increase the throughput of the model regardless of the cost efficiency or pick a node count with a descent throughput, but a low cost. Since there is not perfect scaling of CESM, there exists a particular point in the scaling curve where doubling the number of tasks no longer cuts the run time in half and the cost of core hours of running the model increases. This is where the most cost efficient solution resides.

While running long simulations it may be ideal to pick a target task count where you can achieve the best throughput regardless if it will cost more node hours to run. In this case, it is best to pick any total task count on the tail of the scaling curve and pass this directly to the load balancing tool.

On the other hand, if you are limited on your core hours, you may want to run at the most cost efficient layout for a longer simulation. Because this task count is not known at the run time of the load balancing tool, a new feature was implemented that finds the most cost efficient total task count and provides the user with its optimal layout. This option is also handy for users who are not familiar with the scalability of CESM because it will provide a reliable first guess.

This was implemented as a naive algorithm within the current version of the tool. As a first step, the algorithm picks evenly spaced total task counts that range from the lowest to the highest total task counts that were run within the performance collection step. All of the total task counts are then solved for using the original algorithm, with only the NTASKS_TOTAL and COST_TOTAL values being saved. After all total task counts have been solved for, the timing results are traversed, finding the elbow of the curve. The elbow of the curve is determined to be the most cost efficient task count because it is after this point, the cost of the model begins to increase. This is shown within Figure 6.6.

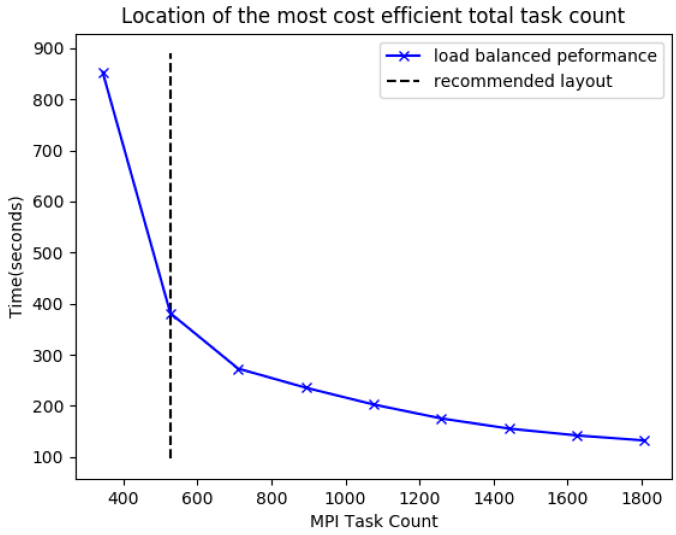


Figure 6.6: The scaling curve of the results from the load balancing tool's total timings are plotted out and the start of the elbow is shown as the most cost efficient solution.

Once the elbow has been found, this becomes the total task target to solve for. It is passed to the solver and the optimal component layout for the most cost efficient solution is then outputted for the user. In Figure 6.6, the most cost efficient task count was estimated at 527 MPI tasks.

6.3 A Comparison of Both Versions of the Load Balancing Tool

Ideally we would have liked to directly compare the results between the MINLP version of the tool versus the MILP version of the tool, but this was not possible. This is because the hardware and software required to run the MINLP version is no longer available. The best comparison that can be made is to compare the reported predicted versus actual total timings from the one degree timings in [1] to the timings that were used in Figure 6.4.

As seen from the timings in Table 6.10, there are large differences in the timing numbers between the MINLP tool verses the MILP tool. The main cause for this difference in the timing numbers are because the timings represent differing simulation lengths. The MINLP tool's timing results captured the total time to simulate 5 days of climate while the MILP tool's timings numbers captured the total time to simulate 10 days of climate. Differences beyond the simulation length are because CESM version 2.0 is more expensive to run than version 1.0. This is because

the components themselves are more expensive to run. By comparing the percentage difference of the actual verses predicted timings from both of the tools, we can better understand the accuracy of each of these versions. The comparison shows that the MILP version of the tool had a difficult time predicting the performance at lower node codes. It was pointed out in Subsection 6.2.1 that this is because the linear solver requires an extrapolated point at one core and this is estimated poorly. At higher node counts, the results between the two versions show roughly similar prediction capabilities.

Table 6.10: The actual versus predicted results from both the MINLP and the MILP version of the load balancing tool. The MINLP tool's timing numbers were from a simulation that simulated 5 days of climate. While the MILP tool's timing numbers were from a simulation that simulated 10 days of climate.

	Total Time in Seconds		**Differences**	
MINLP Tool	**Predicted Timings**	**Actual Timings**	**Actual-Predicted**	**% Difference**
128 Nodes	410.623	425.171	+14.548	3.5%
2048 Nodes	84.484	86.471	+1.987	2.3%
MILP Tool	**Predicted Timings**	**Actual Timings**	**Actual-Predicted**	**% Difference**
432 Nodes	1190.817	1049.787	-141.03	11.8%
864 Nodes	510.716	540.650	+29.934	5.9%
1728 Nodes	314.763	307.096	-7.667	2.4%
2160 Nodes	257.6	263.167	+5.567	2.2%

Chapter 7

Related Work

Load balancing climate models has been of importance for the last several decades. This includes studying both the load balance between components as well as within the components themselves. This section explores contributions by other authors who have worked on this problem.

One of the earlier studies was performed by Foster and Toonen in 1994 [14]. In this work, the authors looked at load balancing the first parallel implementation of an NCAR climate model. The model in which they studied, the Community Climate Model 2(CCM2), is a predecessor of CESM described in this document. This version of the model only contained an atmosphere model [39], therefore, the type of load balancing used would be considered inter component load balancing today.

Load imbalance within the atmosphere model occurs because of the diurnal cycle, where radiation calculations are only performed where it is daylight. In this work, the authors explored a dynamic load balancing approach by reassigning processor ranks at every radiation time step, assigning more resources to where it was daylight. One of the methods explored is similar to a method that is still present in the current version of CAM. In this case, the algorithm swaps data columns with another column at the same latitude, but on the opposite side of the Earth. This creates enough of a load balance of where the radiation calculations are performed and is able to increase the overall performance of the model.

The current version of CAM is much more complex and this type of load balancing only goes so far. A more dynamic approach is needed to fully load balance the atmosphere model, such as increasing the core count over the locations of clouds or for mesh refinement.

In 2009, Sundari, Vadhiyar, and Nanjundiah [40] also looked to improve the performance of atmospheric model, but they used a different dynamic load balancing approach. Their approach would offload expensive calculations done within the atmosphere model to the ranks of other components that were typically idle. This work was performed using the Community Coupled System

Model version 3 (CCSM3), which contained an atmosphere, ocean, sea ice, and land model coupled together. CCSM3 was also a predecessor of CESM.

As the model first starts to run, their algorithms look to identify typical idling periods of the other components. As the model advances, the profiling period switches off and the processor layout of the atmosphere model is switched to use these idle processes from other components. Periodically, the model switches back to a profiling mode and if an imbalance is detected, the atmospheric model's processor layout is altered again. Through this process, the authors stated that they were able to reduce the simulation time by 15%.

In 2007 Koenig and Kale [41] explored the use of dynamic load balancing on the Grid, where computational resources are distributed geographically. In this case, it is critical that the communication latency across resources be minimized in order to achieve better performance results.

The authors developed a load balancing algorithm based on Charm++ and Adaptive MPI that seeks to reduce all communication. The first phase of the algorithm evaluates the amount of communication that will be done by determining the number of messages that will be sent between the different sites. A graph is created based on this communication pattern and a number of partitions are drawn from this graph based on the computation patterns. The second phase looks to load balance across the nodes on the local cluster in which a similar graph is developed and it is partitioned in a similar way.

Keonig and Kale tested their algorithm on the molecular dynamics code, LeanMD. During their test, they used a simulated Grid environment where they introduced artificial latencies that they were able to adjust manually. In all of their tests, their load balancing algorithm was able to reduce the over all cost of running LeanMD. These results look very promising and they mention that it could work on climate codes as well. In this case they suggest running each of the modeling components on separate clusters, but it would take a lot of work to integrate CESM into this type of framework.

In 2013, Kim, Larson, and Chiu explored the use of dynamic load balancing within CESM [42]. They developed the Malleable Model Coupling Toolkit (MMCT), which was an extension

to the MCT used to couple the modeling components within CESM. MMCT looks to minimize the idle time spent by a component waiting for another component within the coupler. This is done by predicting the performance of each component and then adjusting the core counts of each component accordingly.

This method is ideal for a couple of reasons. Even with the creation of the load balancing tool, it is still a static, hands on method, that requires a metric collection step as well as a solver step. It would be ideal if the model could adjust internally and it would ensure that a load balancing step is always performed, potentially saving thousands of node hours be simulation. Second, it is hard to load balancing the model when higher frequency output is desired. An internal dynamic method would make this process easier.

While ideal, the method of dynamically load balancing CESM is not possible. As noted by the authors, this process was simulated in order to test the methods. At this time, core counts within some component models cannot be dynamically allocated. This is because they are determined at compile time, not run time and, therefore, this type of dynamic allocation would require large code changes.

In 2014, Nan, Wei, Xu, Haoyu, and Zhenya [43] developed a load balancing interface called CESMTuner. They used a similar approach as to what was described in the original CESM load balancing tool within this document. The authors' interface first requires the users to collect performance metrics. These metrics are then passed to an MINLP solver which uses similar constraints that were used in our original tool and balance layout is reported back to the user.

Overall they were able to improve the performance greater than what we reported [1], but their results are slightly misleading. In their results, they compared their improvement to a serial execution of CESM, where all components are executed sequentially. As discussed earlier, it is best to run certain components concurrently with other components in order to maximize the parallelization within the coupler calls. This particular component layout has been known to produce the best performance since 2011 [13] and has been the default component configuration for a fully coupled

simulation since that time. Because of the poor choice in their initial layout, they were able to produce a better overall performance improvement.

In 2018, the authors extended the work to include a process layout generator [44]. This work looks to determine which component layout will yield the best performance results. In their example, this new feature confirmed that the optimal layout described in this document would yield the best performance. While this is known to be the optimal layout, this new feature could be helpful in predicting new optimal component layouts when running one of the data components instead of the fully coupled model. In this case, the model reads in prescribed values for a component instead of running the full component. This creates a new load balancing problem in which the model usually runs all of the components sequentially. This extension might be able to point to better configurations for this type of model setup that could increase its performance.

Chapter 8

Future Work

Load imbalance continues to be a problem with ESMs. Somewhere between 1% to 62% of the total time to run ESMs are spent in synchronization calls waiting to couple with other model components [10]. While this work provides a static load balancing method to improve model performance, it would be ideal if the model could be load balanced dynamically as the model is running. This would ensure that long simulations are run with load balanced configuration and performance would be improved.

Dynamic load balancing would be challenging to implement within CESM. This is because certain CESM modeling components require that decompositions be set at compile time and cannot be changed at runtime. This requirement would have to be removed from the ocean and sea ice models before work on dynamic load balancing could be implemented.

This work focused mostly on load balancing between the components, but the individual component models also suffer from load imbalance. Finding the correct decompositions for each model can be tricky because workload is not always predictable. For example, the performance of the physics code in the atmosphere can change if clouds are present in a certain region. Another example is found in the sea ice model where the most expensive calculations are only done where there is sea ice and where there is sunlight. While the decomposition of the sea ice grid is global, calculations are typically done at the North Pole for half the year and then at the South Pole for the other half of the year. Having the ability to dynamically load balance calculations within a particular model would help reduce idle time further.

Chapter 9

Conclusions

CESM is a complex model that presents many computational challenges as we approach exascale systems. Each component model presents different performance challenges and they will need to be addressed separately to meet these challenges. As we look to these types of systems, it will critical to ensure that communication times are kept low and close consideration of memory management will be essential.

In this work we evaluated the performance of CESM, provided a performance analysis in production, and evaluated and validated optimal component layouts for CESM.

While the coupler's computational time is small relative to the model components' run times, it has a potential to slow down the entire CESM if the components are not load balanced. Therefore, when looking to optimize the full model's performance, it is important to verify that the components are balanced and idle times between components are kept to a minimal. This type of optimization is essential now and can save several thousands of core hours for one simulation, but it will be critical as we move to exascale systems.

Load balancing CESM is a difficult task to do and it requires expert knowledge in order to accurately load balance all of the components. The automatic load balancing tool uses this expert knowledge to provide users with a balanced layout. Given a set of constraints, performance data, and a target total core count, it is able to solve the MILP and provide users with predicted run time estimates as well as the amount of resources to give each component to be load balanced within the system.

The MILP version of the tool has the benefit of being more portable and the time to solution is faster than the MINLP version, while predicting performance with the same degree of accuracy for medium to high core counts. As a result of using the tool, we were able to provide the CESM user community with a new layout that has the potential to save thousands of CPU core hours on a popular model configuration.

This work also provided the load balancing tool a new feature, giving it the ability to provide the most cost efficient layout to the user. This is handy if the time to solution is not as important as the overall cost of the simulation or if the user is unfamiliar with the application's scaling and is unsure on a good target task count. This extension evaluates the predicted performance and estimates where the performance starts to slow down and the cost starts to increase.

The load imbalance problems reported in this work exist in all other coupled climate models from around the world. The work done to validate the current constraints by removing the layout constraints from the MILP, allow for the possibility that the tool could be more easily used to load balance other models that do not follow the same layout as CESM. This has the potential to save many millions of core hours world wide by allowing coupled climate models to be more easily balanced before simulations are done.

As we move towards exascale, the model's performance will have to be constantly evaluated and this work is only one piece to help move the CESM in that direction. Being a community code, many people contribute to its development and it is important to educate code contributors about good coding practices for future architectures. We can no longer count on a "free lunch" and expect new generation hardware to increase the model throughput or compilers to optimize our code. It will be hard work optimizing the model for future architectures, but a necessary step to ensure that the model will be able to run efficiently on future platforms.

Bibliography

[1] Yuri Alexeev, Sheri Mickelson, Sven Leyffer, Robert Jacob, Anthony Craig. The heuristic static load balancing algorithm applied to the Community Earth System Model. *2014 IEEE International Parallel and Distributed Processing Symposium Workshops*, pages 1581–1590, 2014.

[2] Sarich. CIME load balancing tool. https://github.com/ESMCI/cime/tree/master/tools/load_balancing_tool, March 2018.

[3] Puma. Climate modelers and the moth. https://www.giss.nasa.gov/research/briefs/puma_02, December 2012.

[4] Gordon E. Moore. Moore's law at 40. *Understanding Moore's Law: Four Decades of Innovation*, pages 67–84, March 2016.

[5] John Dennis, Christopher Kerr, Allison Baker, Brian Dobbins, Kevin Paul, Richard Mills, Sheri Mickelson, Youngsung Kim, Raghu Kumar. Preparing the Community Earth System Model for exascale computing. *Exascale Scientific Applications Scalability and Performance Portability*, pages 207–227, November 2017.

[6] Bryan Lawrence, Michael Rezny, Reinhard Budich, Peter Bauer, Jorg Behrens, Mick Carter, Willem Deconinch, Rupert Ford, Christopher Maynard, Steven Mullerworth, Carlos Osuna, Andrew Porter, Kim Serradell, Sophie Valcke, Nils Wedi, Simon Wilson. Crossing the chasm: how to develop weather and climate models for next generation computers? *Geoscientific Model Development*, pages 1799–1821, May 2018.

[7] James W. Hurrell, M. M. Holland, P. R. Gent, S. Ghan, Jennifer E. Kay, P. J. Kushner, J. -F. Lamrque, W. G. Large, D. Lawrence, K. Lindsay, W. H. Lipscomb, M. C. Long, N. Mahowald, D. R. Marsh, R. B. Neale, P. Rasch, S. Vavrus, M. Vertenstein, D. Bader, W. D. Collins, J. J. Hack, J. Kiehl, S. Marshall. The Community Earth System Model: A framework

for collaborative research. *Bulletin of the American Meteorological Society*, pages 1339–1360, September 2013.

[8] CESM 2.0. http://www.cesm.ucar.edu/models/cesm2/, June 2018.

[9] Mariano Mendez, Fernando G. Tinetti, Jeffrey L. Overbey. Climate models: Challenges for Fortran development tools. *2014 Second International Workshop on Software Engineering for High Performance Computing in Computational Science & Engineering*, pages 6–12, November 2014.

[10] Venkatramani Balaji, Eric Maisonnave, Niki Zadeh, Bryan N. Lawrence, Joachim Biercamp, Uwe Fladrich, Giovanni Aloisio, Rusty Benson, Arnaud Caubel, Jeffrey Durachta, Marie-Alice Foujols, Grenvill Lister, Silvia Macavero, Seth Underwood, Garret Wright. CPMIP: measurements of real computational performance of earth system models in CMIP6. *Geoscientific Model Development*, 10:6–34, January 2017.

[11] Veronika Eyring, Sandrine Boyn, Gerald A. Meehl, Catherine A. Senior, Bjorn Stevens, Ronald J. Stouffer, Karl E. Taylor. Overview of the Coupled Model Intercomparison Project Phase 6 (CMIP6) experimental design and organization. *Geoscience Model Development*, 9:1937–1958, 2016.

[12] Richard B. Neale, Andrew Gettelman, Sungsu Park, Chih-Chieh Chen, Peter H. Lauritzen, David L. Williamson, Andrew J. Conley, Doug Kinnison, Dan Marsh, Anne K. Smith, Francis Vitt, Rolando Garcia, Jean-Francois Lamarque, Mike Mills, Simone Tilmes, High Morrison, Phillip Cameron-Smith, William D. Collins, Michael J. Iacono, Richard C. Easter, Xiaohong Liu, Steven J. Ghan, Phillip J. Rasch, Mark A. Taylor. Description of the NCAR Community Atmosphere Model (CAM 5.0). http://www.cesm.ucar.edu/models/cesm1.0/cam/docs/description/cam5_desc.pdf, November 2012.

[13] Patrick H. Worley, Arthur A. Mirin, Anthony P. Craig, Mark A. Taylor, John M. Dennis, Mariana Vertenstein. Performance of the Community Earth System Model. *Proceedings of*

2011 International Conference for High Performance Computing, Networking, Storage and Analysis, November 2011.

[14] I. T. Foster, B. R. Toonen. Load-balancing algorithms for climate models. *Proceedings of IEEE Scalable High Performance Computing Conference*, pages 674–681, 1994.

[15] Keith Oleson, David Lawrence. Technical description of version 4.5 of the Community Land Model (CLM). http://www.cesm.ucar.edu/models/cesm1.2/clm/CLM45_Tech_Note.pdf, July 2013.

[16] CESM2.0: River runoff models. http://www.cesm.ucar.edu/models/cesm2/river/, June 2018.

[17] R. Smith, P. Jones, B. Briegleb, F. Bryan, G. Danabasoglu, J. Dennis, J. Dukowicz, C. Eden, B. Fox-Kemper, P. Gent, M. Hect, S. Jayne, M. Jochum, W. Large, K. Lindsay, M. Maltud, N. Norton, S. Peacock, M. Vertenstein, S. Yeager. The Parallel Ocean Program (POP) reference manual ocean component of the Community Climate System Model (CCSM) and Community Earth System Model (CESM). http://www.cesm.ucar.edu/models/cesm1.0/pop2/doc/sci/POPRefManual.pdf, 2010.

[18] Ging Li, Adrean Webb, Baylor Fox-Kemper, Anthony Craig, Gokhan Danabasoglu, William G. Large, Mariana Vertenstein. Langmuir mixing effects on global climate: WAVEWATCH III in CESM. *Ocean Modelling*, 103:145–160, July 2016.

[19] David Bailey, Marika Holland, Elizabeth Hunke, Bill Lipscomb, Bruce Briegleb, Cecilia Bitz, Julie Schramm. Community Ice CodE (CICE) users guide version 4.0. http://www.cesm.ucar.edu/models/ccsm4.0/cice/ice_usrdoc.pdf, 2010.

[20] Anthony P. Craig, Sheri A. Mickelson, Elizabeth C. Hunke, David A. Bailey. Improved parallel performance of the CICE model in CESM1. *The International Journal of High Performance Computing Applications*, 29(2):154–165, 2015.

[21] Jim Edwards, John M. Dennis, Mariana Vertenstein, Edward Hartnett. Parallel I/O library (PIO). http://ncar.github.io/ParallelIO/, 2018.

[22] Anthony P. Craig, Mariana Vertenstein, Robert Jacob. A new flexible coupler for earth system modeling developed for CCSM4 and CESM1. *International Journal of High Performance Computing Applications*, pages 26–31, 2012.

[23] CIME documentation. http://esmci.github.io/cime/index.html, June 2018.

[24] Jay Larson, Robert Jacob, Everest Ong. The model coupling toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models. *International Journal of High Performance Computing Applications*, 19(3):277–291, 2005.

[25] Recommendations for the ocean model dynamical core for CESM3. http://www.cesm.ucar.edu/working_groups/Ocean/files/20160505-RFI/recommendations.pdf, May 2016.

[26] Improving vertical mixing parameterizations in MOM6. http://www.cesm.ucar.edu/events/workshops/ws.2018/presentations/omwg/marques.pdf, June 2018.

[27] ESMF: Community infrastructure for building and coupling models. https://www.earthsystemcog.org/projects/esmf/, March 2018.

[28] Richard D. Loft, Aaron H.Anderson, Frank O. Bryan, John M. Dennis, Thomas M. Engel, Pam Gillman, David L. Hart, Irfan Elahi, Siddartha S. Ghosh, Rory Kelly, Anke Kamrath, Gabriele Pfister, Matthias Rempel, R. Justin Small, William Skamarock, Michael Wiltberger, Bryan L. Shader, Po Chen, Benjamin Cash. Yellowstone: A dedicated resource for earth system science. *Contemporary High Performance Computing: From Petascale Toward Exascale*, 2:185–220, 2015.

[29] David L. Hart. NCARs data-centric supercomputing environment yellowstone. http://www.cesm.ucar.edu/management/SSC/Presentations/yellowstone.pdf, November 2011.

[30] David L. Hart. Cheyenne NCARs next-generation data-centric supercomputing environment. https://www2.cisl.ucar.edu/sites/default/files/Cheyenne_Briefing_MesaLab_20160624e.pdf, June 2016.

[31] Integrated performance monitoring. http://ipm-hpc.sourceforge.net/overview.html, October 2009.

[32] Yuri Alexeev, SheriMickelson, Sven Leyffer, Robert Jacob, Anthony Craig. The heuristic static load-balancing algorithm applied to CESM. http://sc13.supercomputing.org/schedule/event_detail.php-evid=post216.html, November 2013.

[33] Robert Fourer, David M. Gay, Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

[34] Leyffer. MINOTAUR: Toolkit for mixed integer nonlinear optimization problems. https://wiki.mcs.anl.gov/minotaur/index.php/Minotaur_Documentation, May 2017.

[35] Roger Fletcher, Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1):327–349, 1994.

[36] Hunke, Lipscomb, Turner, Jeffrey, Elliot. CICE: the Los Alamos sea ice model documentation and software user's manual version 5.1. http://oceans11.lanl.gov/trac/CICE/attachment/wiki/WikiStart/cicedoc.pdf, March 2015.

[37] Prasanna Balaprakash, Yuri Alexeev, Sheri A. Mickelson, Sven Leyffer, Robert Jacob, Anthony Craig. Machine-learning-based load balancing for Community Ice CodE component in CESM. *International Meeting on High Performance Computing for Computational Science*, 2014.

[38] Stuart Mitchell, MichaelO'Sullivan, Iain Dunning. PuLP: A linear programming toolkit for Python. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.416.4985&rep=rep1&type=pdf, September 2011.

[39] The NCAR Community Climate Model. http://www.cgd.ucar.edu/cms/ccm3/history.shtml, June 1999.

[40] Sundari M. Sivagama, Sathish S. Vadhiyar, Ravi S. Nanjundiah. Dynamic component extension: A strategy for performance improvement in multicomponent applications. *The International Journal of High Performance Computing Applications*, 23(1):84–98, February 2009.

[41] Gregory A. Koenig, Laxmikant V. Kale. Optimizing distributed application performance using dynamic grid topology-aware load balancing. *2007 IEEE International Parallel and Distributed Processing Symposium, Long Beach, CA, 2007*, pages 1–10, 2007.

[42] Daihee Kim, J. Walter Larson, Kenneth Chiu. Automatic performance prediction for load-balancing coupled models. *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013.

[43] Ding Nan, Xue Wei, Ji Xu, Xu Haoyu, Song Zhenya. CESMTuner: An auto-tuning framework for the Community Earth System Model. *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst*, pages 282–289, 2014.

[44] Nan Ding, Wei Xue, Zhenya Song, Haohuan Fu, Shiming Xu, Weimin Zhenga. An automatic performance model-based scheduling tool for coupled climate system models. *Journal of Parallel and Distributed Computing*, 2018.

Appendix A

Network Maps

Figures A.1 through A.8 show the communication pattern between the components to the coupler for a two degree finite resolution atmosphere and land grid and a one degree ocean and sea ice grid fully coupled CESM simulation run on 880 MPI ranks. Each of the bubbles indicates a separate MPI task and they are organized into 15 columns because the simulation used 15 MPI ranks per compute node. The bubbles are color coded to indicate the different components. Red bubbles are for the coupler, white for the atmosphere model, green for the land model, purple for the sea ice model, blue for the ocean model, yellow for the river runoff model, black for the glacier model, and aqua for the wave watch model.

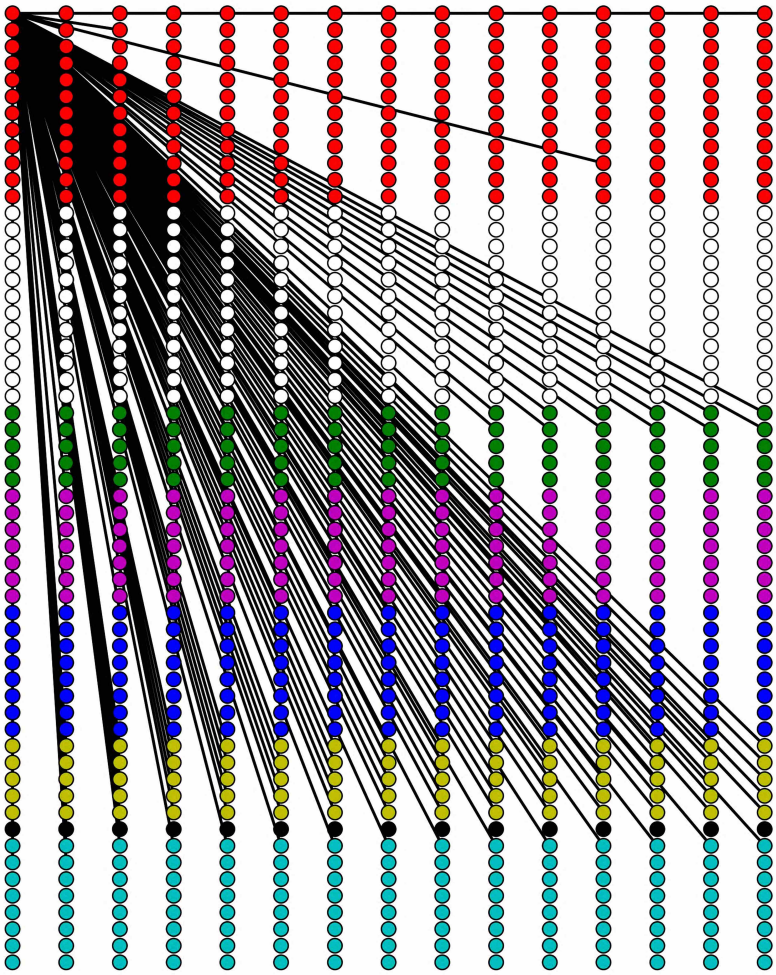


Figure A.1: Communication to the coupler.

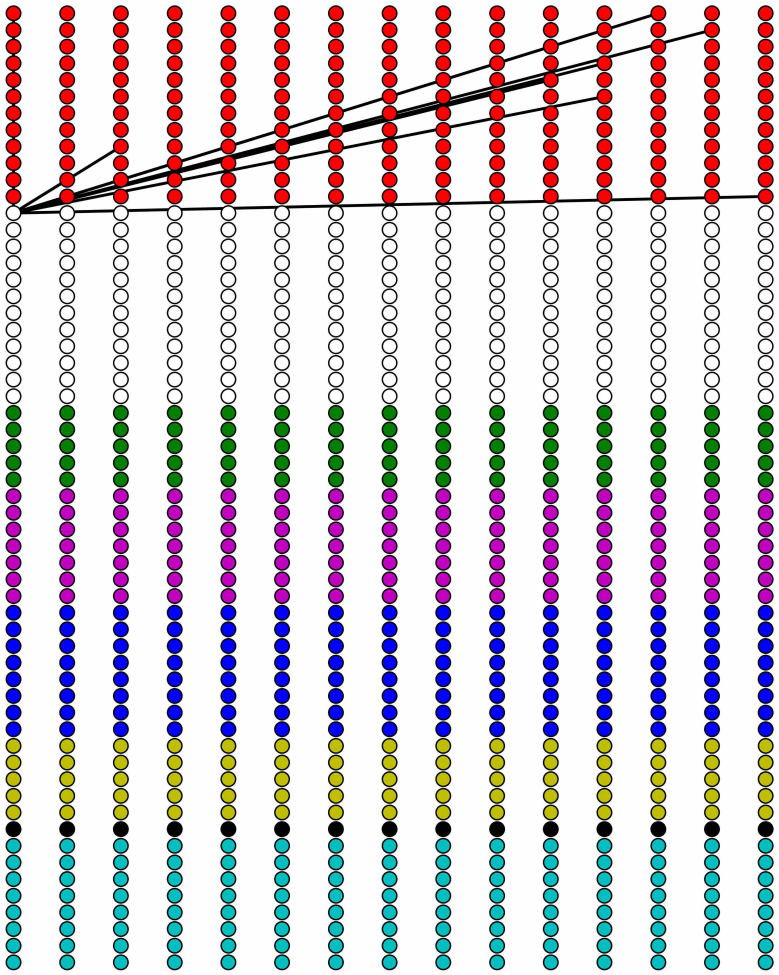


Figure A.2: Atmosphere communication with the coupler.

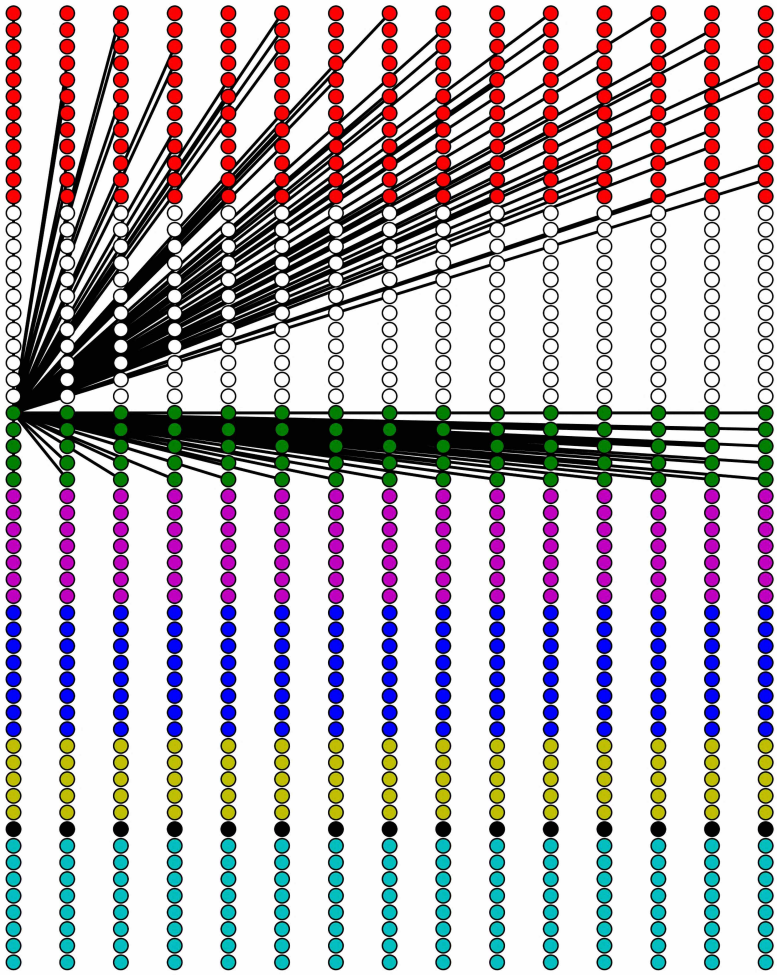


Figure A.3: Land communication with the coupler.

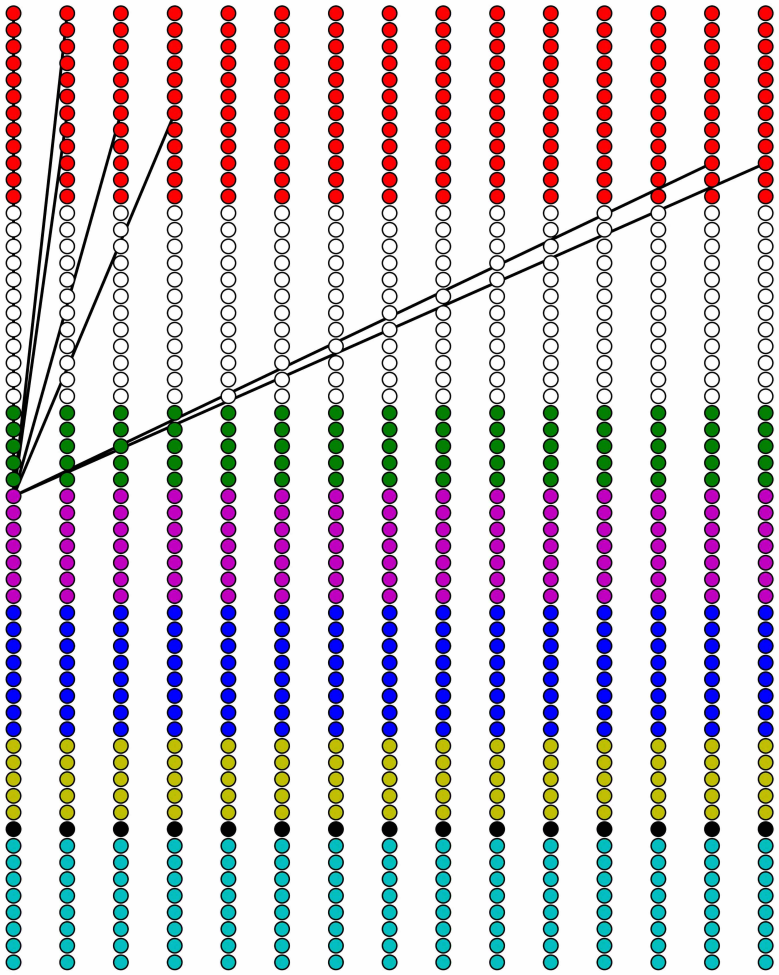


Figure A.4: Sea Ice communication with the coupler.

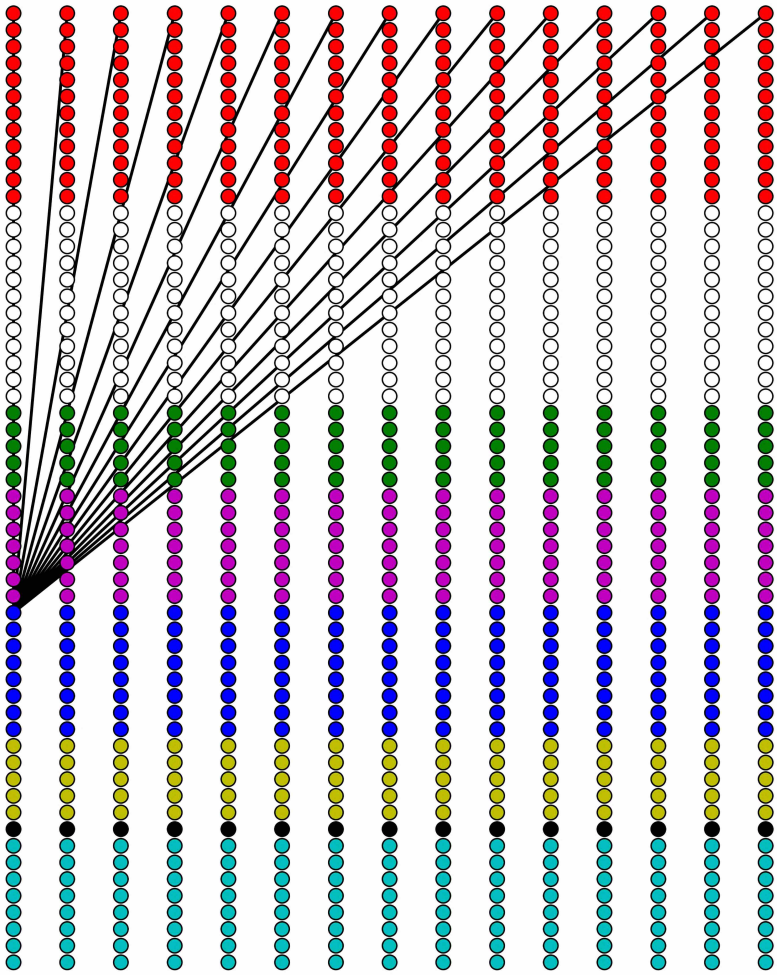


Figure A.5: Ocean communication with the coupler.

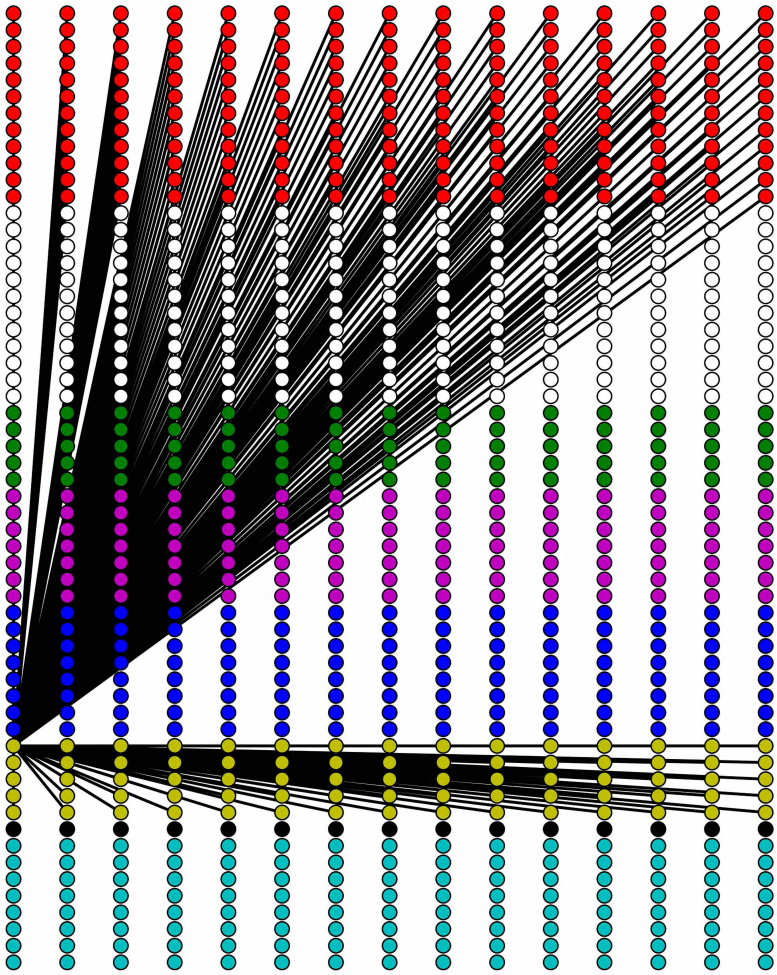


Figure A.6: River Runoff communication with the coupler.

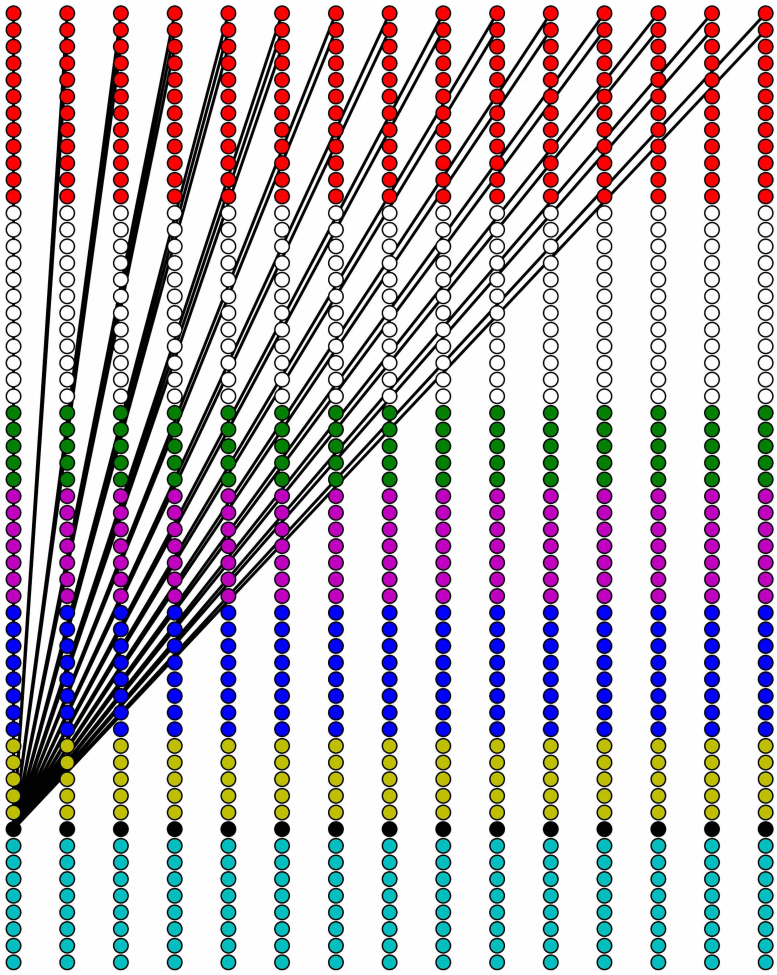


Figure A.7: Glacier model communication with the coupler.

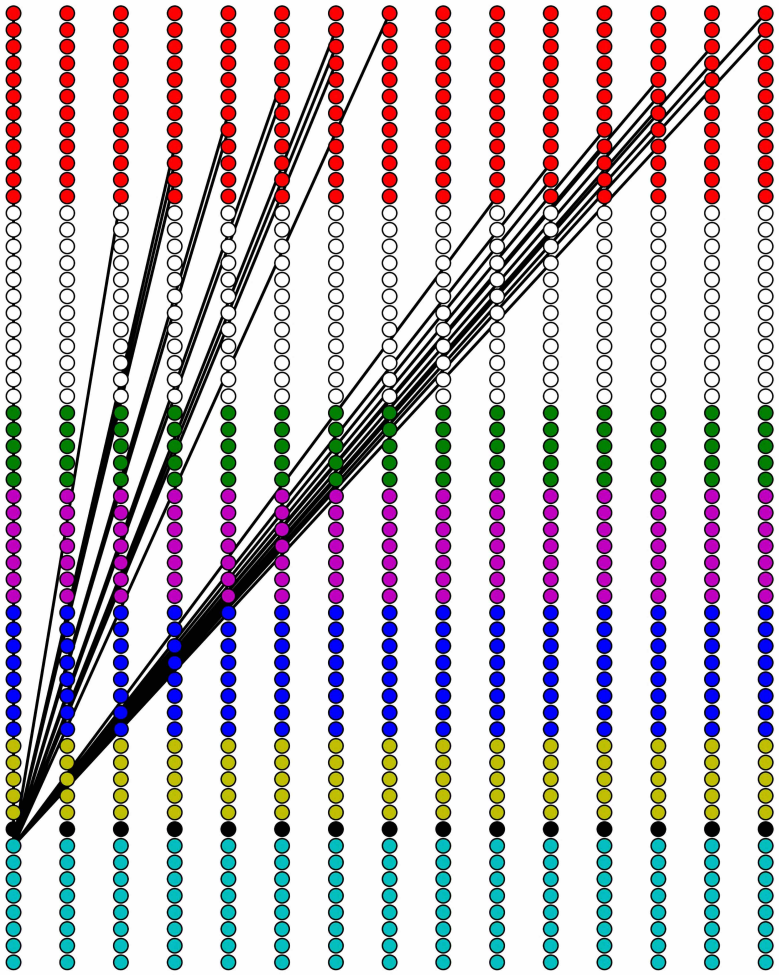


Figure A.8: Wave model communication with the coupler.

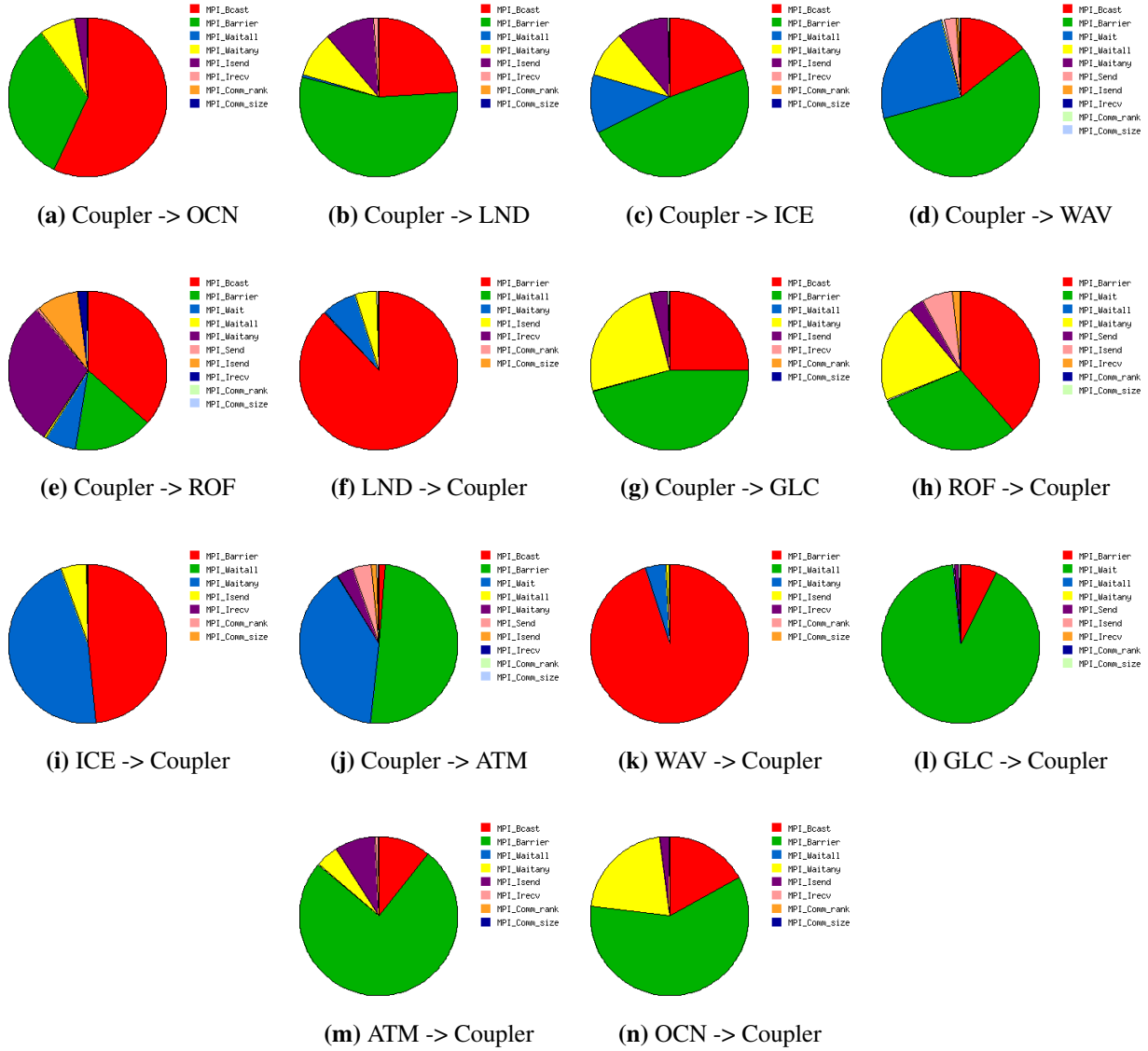
Appendix B

Coupler Timings

The letter codes used in both Table B.1 and Figure B.1 match the labels used in Figure 3.2. It is recommended that Figure 3.2 is used to interpret these results correctly.

Table B.1: The minimum and maximum times in seconds to complete each task for a full day simulated for a 1 degree simulation and a 1/4 degree simulation. The missing high resolution timers for the wave and glacier models were not captured because these components were not available in the version of the model used to collect these timings.The label column in the table matches the labels found within Figure 3.2 in order to add reference to these timings.

		1 Degree Resolution		1/4 Degree Resolution	
Label	**Task**	**Min Time**	**Max Time**	**Min Time**	**Max Time**
1	ATM/OCN Setup	0.001	0.003	0.0	0.001
2	LND Setup	0.004	0.023	0.161	0.316
3	ICE Setup	0.008	0.014	0.103	0.178
4	WAV Setup	0.026	0.045	–	–
5	ROF Setup	0.003	0.005	0.006	0.029
6	LND Post	0.008	0.012	0.029	0.059
7	GLC Setup	0.0	0.0	–	–
8	ROF Post	0.004	0.015	0.015	0.035
9	ICE Post	0.004	0.022	0.025	0.049
10	ATM Setup	0.071	0.099	0.528	0.80
11	WAV Post	0.004	0.01	–	–
12	GLC Post	0.004	0.009	–	–
13	ATM Post	0.006	0.016	0.06	0.09
14	OCN Post	0.002	0.004	0.0	0.0
15	Run ICE	1.463	1.591	1.204	4.125
16	Run LND	0.951	1.447	30.62	32.97
17	Run ROF	0.017	0.12	0.904	1.162
18	Run WAV	0.77	1.06	–	–
19	Run OCN	11.84	11.915	8.483	8.564
20	Run ATM	9.586	10.289	122.963	137.53
21	Run GLC	0.006	0.025	–	–
a	Coupler->OCN	0.007	0.028	0.0	0.002
b	Coupler->LND	0.009	0.013	0.109	0.16
c	Coupler->ICE	0.015	0.033	0.115	0.229
d	Coupler->WAV	0.011	0.021	–	–
e	Coupler->ROF	0.007	0.011	0.019	0.035
f	LND->Coupler	0.183	1.066	0.223	0.409
g	Coupler->GLC	0.0	0.0	–	–
h	ROF->Coupler	0.004	0.009	0.004	0.012
i	ICE->Coupler	0.012	0.031	0.049	0.096
j	Coupler->ATM	0.024	0.047	0.223	0.488
k	WAV->Coupler	0.24	0.929	–	–
l	GLC->Coupler	0.0	0.0	–	–
m	ATM->Coupler	0.013	0.029	6.535	21.114
n	OCN->Coupler	0.404	1.199	0.0	78.504



(a) Coupler -> OCN **(b)** Coupler -> LND **(c)** Coupler -> ICE **(d)** Coupler -> WAV

(e) Coupler -> ROF **(f)** LND -> Coupler **(g)** Coupler -> GLC **(h)** ROF -> Coupler

(i) ICE -> Coupler **(j)** Coupler -> ATM **(k)** WAV -> Coupler **(l)** GLC -> Coupler

(m) ATM -> Coupler **(n)** OCN -> Coupler

Figure B.1: The proportion of time spent in each of the communication steps for the 1 degree simulation. The letter of each subplot corresponds to lettering scheme in Figure 3.2.